

NAME

clifm – The Command Line File Manager

SYNOPSIS

clifm [*OPTION*]... [*PATH*]

INDEX

1. Getting help
2. Description
3. Parameters
 - . Positional parameters
 - . Options
4. Commands
5. File Filters (by filename, file type, and MIME type)
6. Keyboard shortcuts
7. Theming
8. Builtin expansions
9. File opener (third-party openers are also supported)
10. Shotgun, a builtin files previewer
11. Auto-suggestions (including a warning prompt for invalid command names)
12. Shell functions
13. Plugins
14. Autocommands
15. File tags
16. Virtual directories
17. Note on speed
18. Kangaroo frequency algorithm
19. Environment
20. Security
21. Miscellaneous notes
22. Files
23. Examples

1. GETTING HELP

There are several ways of getting help in **clifm**. Once in the program, enter `?` or `help` for 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 jump to the **KEYBOARD SHORTCUTS** section.

To get help about some specific topic, type `help <TAB>` to get a list of available help topics. Choose the topic you want and then press `Enter`.

For a list of available commands and a brief description type `cmd<TAB>`.

Help for all internal commands can be accessed via the `-h` or `--help` flags. For example, to get help about the selection function, `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`). Enter `ih` to run the plugin (it depends on **fzf**(1)) 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 interaction 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/**E**xecute 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. PARAMETERS

POSITIONAL PARAMETERS

If the first non-option parameter is a directory, **clifm** will start in this directory (for example, the command `clifm /etc` instructs **clifm** to start in the directory `/etc`).

If not specified, the first workspace is used. To start in a different workspace use the `-w` option (for instance, `clifm -w4 /etc`).

If no positional parameter, and the `-w` option is not used, **clifm** starts by default in the last visited directory (and in the last active workspace). To disable this behavior use `--no-restore-last-path` (see below).

OPTIONS

Note: If compiled in POSIX mode, the following list of options does not apply. In this case, run `clifm -h` to get the actual list of options. (To make sure run `clifm -v`: if compiled in POSIX mode the version number is followed by "-POSIX").

-a, --show-hidden
do not ignore entries starting with `.`

-A, --no-hidden
ignore entries starting with `.`

- b, --bookmarks-file=FILE**
set an alternative bookmarks file
- c, --config-file=FILE**
set an alternative configuration file
- D, --config-dir=DIR**
set an alternative configuration directory (if configuration files do not exist already, they will be created in *DIR*).
- e, --no-eln**
do not print ELNs (entry list number) at the left of filenames. Bear in mind, however, that though ELNs are not printed, they are still there and can be used as always.
- E, --eln-use-workspace-color**
ELNs use the current workspace color
- f, --dirs-first**
list directories first
- F, --no-dirs-first**
do not list directories first
- g, --pager**
enable *Mas*, the builtin pager for file listing
- G, --no-pager**
disable the files pager
- h, --help**
show this help and exit
- H, --horizontal-list**
list files horizontally (instead of vertically)
- i, --no-case-sensitive**
ignore case distinctions when listing files
- I, --case-sensitive**
do not ignore case distinctions when listing files
- k, --keybindings-file=FILE**
set an alternative keybindings file
- l, --long-view**
print file extended metadata next to filenames. Displayed fields can be customized via *--prop-fields* (*PropFields* in the configuration file). Set a custom time/date format using *--time-style* (*TimeStyle* in the configuration file).
- L, --follow-symlinks-long**
when running in long view, show information for the file symbolic links reference rather than for the symbolic links themselves
- m, --dirhist-map**
enable the directory history map to keep in view previous, current, and next entries in the directory history list
- o, --autols**
the *cd* command changes directory **and** lists files automatically
- O, --no-autols**
the internal *cd* command works like the shell *cd* command: change directory but **do not** list files automatically

- P, --profile=PROFILE**
use *PROFILE* as profile. If *PROFILE* does not exist, it will be created. The default profile is *default*.
- r, --no-refresh-on-empty-line**
do not refresh the current list of files when pressing *Enter* on an empty line
- s, --splash**
enable the splash screen
- S, --stealth-mode**
in stealth mode (also known as incognito or private mode) no trace is left on the host system. Nothing is read from files nor any file is created: all settings are set to the default values. However, most settings can still be controlled via command line options and 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 --full-dir-size --no-dirs-first*. The total size of the current directory will be printed after the list of files. Press *Ctrl+Alt+i* (or *Alt+Tab*) to toggle this mode in-place.
- T, --trash-dir=DIR**
set an alternative trash directory
- 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 to start in workspace *NUM* and in path *PATH*. E.g.: *clifm -w4 /etc*.
- 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* is one of: 0 = none, 1 = name, 2 = size, 3 = atime, 4 = btime, 5 = ctime, 6 = mtime, 7 = version, 8 = extension, 9 = inode, 10 = owner, 11 = group, 12 = blocks, 13 = links, or 14 = type. Both numbers and names are allowed. E.g: *--sort=9* or *--sort=inode*.
- bell=TYPE**
set the terminal bell type, where *TYPE* is: 0 = none, 1 = audible, 2 = visible (requires readline >= 8.1), and 3 = flash. Defaults to **1 (visible)**, and, if not available, **0 (none)**. Only numbers are allowed.
- 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 the last visited directory to *\$XDG_CONFIG_HOME/clifm/.last* to be later accessed by the corresponding shell function at program exit. Consult the **SHELL FUNCTIONS** section below for more information.

- color-scheme=NAME**
use the color scheme named *NAME*
- color-links-as-target**
colorize symbolic links using the target file color
- cwd-in-title**
print the current working directory in the terminal window title (otherwise, only the program name is printed)
- data-dir=DIR**
load data files, such as plugins, color schemes, and default configuration files, from *DIR*
- desktop-notifications[=STYLE]**
set the desktop notifications style. Supported values: *kitty* (requires the Kitty terminal or a terminal supporting the Kitty Notifications Protocol), *system*, or *false*. If *STYLE* is omitted, it defaults to *system*. Enter *help desktop-notifications* in **clifm** for more information.
- disk-usage**
show disk usage for the filesystem where the current directory resides (in the form *free/total (free-percentage) fs-type dev-name*)
- full-dir-size**
when running in long view, print the full (or total) size of directories
- fuzzy-algo=VER**
fuzzy matching algorithm, where *VER* is either *1* (faster, but not Unicode aware), or *2* (slower, Unicode aware). Bear in mind however that the second algorithm will fallback to the first one (because it's faster) whenever the query string contains only ASCII characters, to minimize thus the performance penalty.
- fuzzy-matching**
enable fuzzy matches for filename/path completions and suggestions
- fzfpreview-hidden**
enable file previews for tab completion (fzf mode only) with the preview window hidden (toggle it via *Alt+p*)
- fzf~~tab~~**
use *fzf* to display completion matches
- fnftab**
use *fnf* to display completion matches
- icons**
enable icons
- icons-use-file-color**
instead of a specific color, icons take the color of the corresponding filename (specified either via file type or file extension). Useful when building custom color schemes. This option implies **—icons**. Only if compiled with support for either icons-in-terminal or Nerdfonts. The default build is compiled with emoji-icons support, in which case this option is ignored (Unicode icons have their own color builtin)
- int-vars**
allow the use of internal variables (e.g.: *VAR=/bin; cd \$VAR*)
- list-and-quit**
list files and quit. Useful in conjunction with positional parameters. E.g.: *´clifm —list-and-quit /etc´*
- ls** short for *—list-and-quit*

--lscolors

read file colors from the **LS_COLORS** environment variable (BSD **LSCOLORS** is also supported). Bear in mind that clifm-specific colors (like empty directory or inaccessible file) will be disabled. Note also that colors for specific filenames, as defined in **LS_COLORS**, are not supported. Consult **dircolors(1)** for more information about **LS_COLORS**, or the **ls(1)** BSD manpage in the case of **LSCOLORS**.

--max-dirhist

maximum number of visited directories to remember

--max-files=NUM

list only up to *NUM* files. Use *-1* or *'unset'* to remove this limit (default). See the *mf* command for a more detailed description.

--mimelist-file=FILE

set *FILE* as Lira's configuration file. See the **FILE OPENER** section below for more information

--mnt-udisks2

use *udisks2* instead of *udev* (default), for the *media* command

--no-apparent-size

print file sizes as used blocks (actual device usage) instead of used bytes (apparent size). Equivalent to *--physical-size*.

--no-bold

disable bold colors (applies to all color schemes)

--no-cd-auto

by default, **clifm** changes directories by just specifying the corresponding ELN (e.g. *'12'* instead of *'cd 12'*). This option forces the use of *cd*.

--no-classify

by default, **clifm** appends a file type indicator character to filenames when running with no colors (see the **--no-color** option below) and a directory indicator (plus a files counter) when running with colors. Classification characters are:

/n: directory (n = files counter)

@: symbolic link

!: broken symbolic link

|: fifo/pipe

=: socket

*: executable file

+: block device

-: character device

?: unknown file type Use this option to disable file type classification. Note that this option disables the files counter too.

--no-clear-screen

do not clear the screen before listing files

--no-color

disable colors

--no-columns

disable columns when listing files (use a single column)

--no-dir-jumper

disable the directory jumper function

--no-file-cap

do not check file capabilities when listing files (only meaningful for performance reasons)

- no-file-ext**
do not check file extensions (mostly used to colorize specific filenames) when listing files
- no-files-counter**
disable the files counter for directories (speeding up thus the listing process: counting files in directories is particularly expensive).
- no-follow-symlinks**
do not follow symbolic links when listing files. Overrides both *-L* and *--color-links-as-target*.
- no-fzfpreview**
disable file previews for tab completion (fzf mode only)
- no-highlight**
disable syntax highlighting. To customize highlighting colors see the **COLOR CODES** section below.
- no-history**
do not write commands to the history file (see also the *HistIgnore* option in the configuration file)
- no-open-auto**
same as *--no-cd-auto*, but for files instead of directories
- no-refresh-on-resize**
do not attempt to refresh the list of files upon window's resize
- 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-truncate-names**
do not truncate filenames (see *MaxFilenameLen* in the configuration file)
- no-unicode**
do not use Unicode decorations
- no-warning-prompt**
disable the warning prompt (used to highlight invalid command names)
- no-welcome-message**
disable the welcome message
- only-dirs**
list directories only
- open=FILE**
run as a standalone file opener: open *FILE* and exit, where *FILE* can 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**
use *APPLICATION* (e.g.: *rifle* or *xdg-open*) as files opener/launcher (instead of *Lira*, **clifm**'s default opener).
- pager-view=MODE**
list files in the pager according to *MODE*. Possible values: *auto* (use the current listing mode - this is the default), *long* (list files in long view), and *short* (list files in short view).

- physical-size**
same as **--no-apparent-size**
- preview=FILE**
display a preview of *FILE* (via *Shotgun*) and exit. Use **--shotgun-file** to set an alternative configuration file via. Consult the **SHOTGUN** section below for more information.
- print-sel**
print the list of selected files. The maximum number of selected files to be printed can 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.
- prop-fields=FORMAT**
set fields to be displayed in long view mode. For information on how to construct this format string consult the *PropFields* option in the configuration file.
- ptime-style=STYLE**
time/date style used by the *p/pp* command and the **--stat/--stat-full** command line switches. Available styles: **default**, **iso**, **long-iso**, **full-iso**, **full-iso-nano**, **+FORMAT**. **FORMAT** is interpreted like in **strftime(3)**. Nano-second precision is available via the **%N** modifier (like in **date(1)**).
- readonly**
run in read-only mode (internal commands able to modify the filesystem are disabled). Disabled commands are: *ac*, *ad*, *bb*, *bl/bleach*, *br/bulk*, *c*, *dup*, *l*, *le*, *m*, *md*, *n/new*, *oc*, *paste*, *pc*, *r*, *rr*, *t/trash*, *tag*, *te*, *u/undel/untrash*, and *vv*, plus the shell commands *cp*, *rm*, *mv*, *ln*, *mkdir*, *rmdir*, *link*, and *un-link*.
- report-cwd**
report the current directory to the underlying terminal (via the OSC-7 escape sequence, not supported by all terminals)
- 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 (**--secure-env** is implied). 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
- sel-file=FILE**
set *FILE* as selections file
- share-selbox**
make the Selection Box common (that is, accessible) to different profiles. By default, each profile has a private Selection Box.
- shotgun-file=FILE**
set *FILE* as *shotgun*'s configuration file. See the **SHOTGUN** section below for more information
- si** print file sizes in powers of 1000, as defined by the International System of Units (SI), instead of 1024
- smenutab**
use *smenu* to display completion matches
- sort-reverse**
sort files in reverse order (e.g.: *z-a* instead of *a-z*)

- stat FILE...**
run the *p* command on FILE(s) and exit. This must be the **last option** on the command line. Use **--ptime=style** to set a custom date/time format.
- stat=full FILE..**
same as **--stat**, but it runs the *pp* command (instead of *p*) on FILE(s)
- stdtab**
use the standard mode (readline's builtin) for tab completion
- time=style=STYLE**
time/date style used in long view. Available styles: **default**, **relative**, **iso**, **long-iso**, **full-iso**, **+FORMAT**. FORMAT is interpreted like in **strftime(3)**.
- trash-as-rm**
the *r* command executes the builtin *trash* (see the *t* command) instead of **rm(1)** to prevent accidental deletions
- unicode**
Unicode decorations are used provided Unicode support is detected for the running terminal. Use this option to force Unicode decorations.
- virtual-dir=PATH**
use PATH as **Clifm**'s virtual directory
- virtual-dir=full-paths**
print filenames in virtual directories as full paths instead of just base names
- vt100**
run in VT100 compatibility mode

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

4. COMMANDS

Help for all commands listed here can be accessed via the **-h** or **--help** options. For example: *p --help* to get help about the properties function.

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 is named "2", then **clifm** cannot know in advance if the command refers to the ELN 2 or to the filename "2". In we want the ELN, we just write the ELN number, for example: *s 2*. But if we want the filename, we need to escape the filename using the backslash character: *s \2*.

NOTE 3: **clifm** supports **fused parameters** for internal commands taking an ELN or range of ELNs 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 ELNs will not 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 filename 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*. ELNs, of course, are allowed. Example: 12.

/PATTERN [-filetype] [-x] [DIR]

this is the **quick search** function. 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.

You can list previously used search patterns via TAB: */*<TAB>*.

Note: By default, the search function attempts to resolve a pattern first as glob, and then, if no matches are found, as a regular expression. This behavior can be customized however in the configuration file, via the *SearchStrategy* option.

Note 2: If no further parameter is provided, but only a glob pattern (wildcards), you can expand the pattern into the corresponding matches via the TAB key. For example, to list all C files in the current directory: */*.c<TAB>*.

Note 3: Expressions containing no pattern metacharacter are automatically transformed into a glob/regular expression (depending on the value of the *SearchStrategy* option). For example, */test* becomes **test** or */*.test.**.

1. Case sensitivity

By default, regular expressions 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.

2. Destination directory

To search for files in any directory other than the current directory, specify the directory name as a further parameter. This parameter (DIR) can 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.

3. File type filter

The result of the search can be further filtered by specifying a filter type: *-b*, *-c*, *-d*, *-f*, *-l*, *-p*, *-s*, *-O*, and *-P* (block device, character device, directory, regular file, symbolic link, FIFO/pipe, socket, door (Solaris), and port (Solaris) 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*.

4. Negate a pattern

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'.

5. Recursive search

To perform a recursive search use the `-x` parameter, and, optionally, a search path (DIR) (file type filter is not allowed). The search will be performed using **find**(1) as follows: *find DIR MODE PATTERN*. If no search path is provided, the search is executed starting in the current directory. Otherwise, the search starts in DIR. MODE is one of:

- `-name`: if *SearchStrategy* is not *regex-only* and *CaseSensitiveSearch* is set to *true*
- `-iname`: if *SearchStrategy* is not *regex-only* and *CaseSensitiveSearch* is set to *false*
- `-regex`: if *SearchStrategy* is *regex-only* and *CaseSensitiveSearch* is set to *true*
- `-iregex`: if *SearchStrategy* is *regex-only* and *CaseSensitiveSearch* is set to *false*

;[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...

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 filenames, 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 archive 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. If set to on, *DIR* amounts to *cd DIR*.

actions [*list*] [*edit*] [*APP*]

to list available actions (or plugins) use the *list* subcommand. Note that, since *list* is the default action, it can be omitted.

Use the *edit* subcommand to add, remove or modify custom actions (using APP if specified or the default associated application for text files 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) edit the actions file (via *actions edit*) and 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). Example: "myaction=myscript.sh". b) Now, drop the corresponding script (in our example, *myscript.sh*) into the plugins directory, usually, `~/config/clifm/plugins` (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.

Note that all arguments passed to the action command (*myaction*) are passed to the script or program as well (*myscript.sh*), which is executed via the system shell.

To assist the user when writing plugins, **clifm**'s state information is exported via environment variables when running plugins. For example, **CLIFM_LONG_VIEW** is set to 1 if currently running in long view. See the **ENVIRONMENT** section for the complete list of exported values.

The plugins bundled with **clifm** (take a look at the plugins directory) can be used as a starting point to create new 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 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 (i.e. the POSIX syntax for the *alias* shell command), 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 filesystem, 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 can 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 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.

auto [*list* | *none* | *unset* | *OPTION=VALUE...*]

Set a temporary autocommand for the current directory.

Unlike **permanent** autocommands, defined in the configuration file via the *autocmd* keyword (see the **AUTOCOMMANDS** section below), options set via the *auto* command are **temporary**, i.e., valid only for the current directory and the current session.

Options set via this command take precedence over both permanent autocommands and regular options (set either via the command line or the configuration file).

Examples

List available autocommands

auto list

List files in the current directory in long view

auto lv=1

List only PDF files, set the color scheme to nord, and sort files by size

auto ft=*.pdf\$,cs=nord,st=size

The same list of options can be specified sequentially (i.e., previous options are preserved)

auto ft=*.pdf\$

auto cs=nord

auto st=size

Unset the files filter and the color scheme, and change sort to blocks

auto ft=,cs=,st=blocks

Unset all temporary autocommands previously set for the current directory

auto unset

Reload the current directory ignoring all autocommands (including permanent autocommands)

auto none

For the list of available option codes consult the **AUTOCOMMANDS** section or enter *help auto-commands*.

ao, auto-open [*on, off, status*]

toggle the auto-open function. If set to on, *FILE* amounts to *open FILE*.

b, back [*h, hist*] [*clear*] [*W!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 (to prevent a directory from being added to the directory history list use the *DirhistIgnore* option in the main configuration file). You can see this list by typing *b hist* or *b h*, 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 highlighted line (by default printed in bold cyan) 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. You can take a look at the *dh* command as well.

bb, bleach *ELN/FILE...*

Bleach is a builtin filenames cleaner (based on detox [<https://github.com/dharple/detox>]), whose main aim is to rename filenames using only safe characters. Bleach cleans up filenames 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 filenames:

- NUL (\0) and slash (/) characters are completely disallowed
- Only characters from the **P ortable 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 (see the *cleaner_table.h* in the source code).
- Filenames never start with a dash (–)
- Files named . and .. are not allowed
- Append .bleach to single character filenames
- Do not let a replacement filename start with a dot (hidden) if the original does not
- Max filename length is NAME_MAX (usually 255)

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

If the replacement filename already exists, a dash and a number (starting from 1) will be appended. E.g.: 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 this 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 this 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 can match any part of a directory name: *bd er*, for instance, will take you to */home/user*, since it is an unambiguous query.

bl FILE...

Create symbolic links (in the current directory) for each specified file. For example, to create symbolic links in the directory *dir* for all PNG files in the current directory, issue these commands: *s *.png, cd dir*, and then *bl sel*.

bm, bookmarks [*a*, *add* FILENAME NAME [SHORTCUT]] [*d*, *del* NAME] [*e*, *edit* [APP]] [NAME, SHORTCUT]

Bookmarks can be managed either from the bookmark manager screen or from the command line.

1. The bookmark manager screen

To access the bookmark manager screen simply enter *bm*. Here you can *cd* to the desired bookmark by entering either its ELN or name (regular files can be bookmarked as well). In this screen you can also add, remove, or edit your bookmarks by simply entering 'e' to edit the bookmarks file (which is simply a list of lines with this format: *name:path*. E.g.: "docs:/home/user/documents"). Make your changes, save, and exit.

2. The command line

Command	Description
<code>bm add /media/mount mnt</code>	Bookmark the <i>/media/mount</i> directory as "mnt"
<code>bm mnt</code>	Change to/open the bookmark named "mnt"
<code>bm del mnt</code>	Delete the bookmark named "mnt"
<code>bm edit</code>	Edit your bookmarks

A handy use for the bookmarks function is to create bookmarks using short names, which will be later easily accessible via tab completion.

The *b:* construct

The *b:* construct is used as a way to quickly access/operate on bookmarks. A few examples:

Command	Description
<code>b:<TAB></code>	List available bookmarks
<code>b:net</code>	Change to the bookmark named "net" (1)
<code>p b:bm1 b:bm2</code>	Print file properties of the bookmarks named "bm1" and "bm2"
<code>s b:</code>	Select all bookmarks at once

(1) If you are not sure about where a bookmark points to, type *b:NAME<TAB>*.

br, bulk *ELN/FILE...* [:EDITOR]

rename at once all files passed as arguments to the function. It accepts single and multiple filenames, wildcards, ELN ranges, and the *sel* keyword. Example: *br myfile 4-10 sel*.

Each filename will be copied to a temporary file, which will be opened via EDITOR (default associated application for plain text files if omitted), 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 for confirmation.

This builtin bulk rename function won't deal with deletions, replacements, filename conflicts and the like. For a smarter alternative use **qmv**(1).

c, m, md, r

short for the following commands respectively: *cp -iRp*, *mv -i*, *mkdir -p*, and *rm* (for files) or *rm -r* (for directories).

By default, the *c*, *m*, and *r* commands ask for confirmation before operations. Since this might sometimes be quite intrusive (specially when operating on large amount of files), it is possible to turn interactivity off in two different ways:

a) For the current command only: via the *-f*, *--force* switch. Example: *c -f sel*, *m -f sel*, or *r -f **.

b) Permanently. Use the *cpCmd*, *mvCmd*, and *rmForce* options in the configuration file to permanently set any of these commands to non-interactive mode.

To use these commands without any of these arguments, or with any other argument you want, use the non-abbreviated (shell) command, for instance, *cp* instead of *c*. Of course, you can also create aliases to use your preferred commands, for example, "*c='cp -adp'*". Consult the *alias* command above for more information.

The *l* command allows the use of the *e*, *edit* option to modify the destination of a symbolic link. Example: *l edit 12* (or *le 12*) to relink the symbolic link corresponding to the file whose ELN is 12.

When using the *sel* keyword and no destination is provided, *c* and *m* will copy/move selected files to the current directory.

Whenever *sel* is not used, but just a source filename (and no destination 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 filename as base, so that it does not need to be typed twice. For this alternative prompt, only tab completion for filenames is available.

Clifm supports **advcp**(1), **wcp**, and **rsync**(1) to copy files (they include a progress bar). To use them instead of **cp**(1) set the corresponding option (*cpCmd*) in the configuration file. If *advcp* is selected, the command used is *advcp -giRp* (or *advcp -gRp*, for non-interactive mode). If *rsync*, the command is *rsync -avP*. *wcp* takes no argument.

advmv(1) is also supported to move files (to add a progress bar to the move command). Use the *mvCmd* option in the configuration file to choose this alternative implementation of *mv*. In this case, the command used is *advmv -gi* (or *advmv -g* for non-interactive mode).

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 to this 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 to it.
4. Check directories in the current working directory. If a matching directory is found, change to it.

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

Unlike the shell **cd**(1) command, **clifm**'s builtin *cd* function not only changes the current directory, but also lists its content (provided the option *AutoLs* 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 file listing can be disabled by either setting *AutoLs* to "false" in the configuration file or running **clifm** with the *-O* or *--no-autols* option.

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

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.

colors preview the current color scheme

config [*edit [APP]*] [*reset, reload, dump*]
Manage the main configuration file.

To edit the configuration file use the *edit* subcommand. If an application is specified (*config edit APP*), *APP* will be used to open the file (otherwise, the default associated program will be used). Edit settings to your liking if necessary, save, and quit the editor. Changes are automatically applied. Note that, since *edit* is the default action, it can be omitted. Enter just *config* to open the configuration file, or *config APP* to open it using *APP*.

Use the *reload* subcommand to reload the main configuration file and update settings accordingly.

Use the *reset* subcommand to generate a fresh configuration file and create a backup copy of the old one (named *clifmrc.YYYYMMDD@HH:MM:SS*).

The *dump* subcommand prints the list of settings (as defined in the main configuration file) with their current value. Those differing from the default values are highlighted, and the default value for the corresponding option is displayed in brackets.

cs, colorschemes [*edit [APP]*] [*n, name*] [*p, preview*] [*check-ext*] [*NAME*]
with no arguments, list available color schemes (*cs name* (or *cs n*) to print the current color scheme name).

To get a preview of the current color scheme use the *preview* subcommand: *cs preview*.

Use the *check-ext* subcommand to check for file extension conflicts: *cs check-ext*.

Use the *edit* subcommand to open/edit the configuration file of the current color scheme (open with *APP* if specified or via the default associated application).

To switch color schemes, specify the color scheme name: *cs NAME*. (Tab completion is available: *cs <TAB>*).

d, dup *FILE...*

Duplicate files passed as parameters, either directories or regular files. The user will be asked for a destination directory. Duplicated filenames are generated by appending ".copy" to the basename of each source file. For example: *d /my/file* will copy */my/file* to the directory selected by the user as *file.copy*. If *file.copy* already exists, an extra suffix will be added as follows: *file.copy-N*, where *N* is a positive integer (starting at 1).

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

dh [*STRING*] [*PATH*] [*!ELN*]

With no parameters, it prints the directory history list. To filter this list just pass a query string: only entries matching this query will be displayed. In both cases, tab completion is available. For example: *dh down<TAB>* will list only those entries matching *down* (fuzzily, if *fuzzy-matching* is enabled).

To access a specific entry, you can pass the entry number preceded by an exclamation mark. For

example, if you want the entry number 12, enter *dh !12* to change to the corresponding directory.

Finally, if an absolute path is passed as first parameter, *dh* works just as the *cd* command.

Note: Take a look at the *j* command as well. Both commands deal with the list of visited directories, but in slightly different ways: while *dh* deals with the list of the last *MaxDirhist* entries (see the configuration file), the *j* command deals with the *ranked* list of visited directories.

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. E.g.: *ds **.

You can also pass the filename(s) (or ELNs) to be deselected as a parameter. For example: *ds my-file 24*.

Tab completion is available for this command.

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 can be directories, filenames, ELNs or some search expression like *"*.c"*.

ext [*on*, *off*, *status*]

toggle external commands.

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*, and *f !ELN*.

fc [*on*, *off*, *status*]

By default, **clifm** prints the amount of files contained by listed directories next to the directory 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* subcommand to disable it. To permanently disable it, use the *FilesCounter* option in the configuration file.

ff, dirs-first [*on*, *off*, *status*]

toggle list directories first.

ft, filter [*unset*] [*[/]REGEX,=FILE-TYPE-CHAR*]

filter the current list of files, either by filename (via a regular expression) or file type (via a file type character).

With no argument, *ft* prints the current filter. To remove the current filter use the *unset* option. To set a new filter enter *ft* followed by a filter expression (use the exclamation mark to reverse the meaning of a filter). Examples:

Exclude hidden files:

ft !^.

List only files ending with .pdf:

ft .*\.pdf\$

List only symbolic links:

ft =l

Exclude socket files:

ft !=s

The list of file type characters is included in the **FILE FILTERS** section below.

The filter will be lost at program exit. To permanently set a filter use the *Filter* option (in the configuration file) or the **CLIFM_FILTER** environment variable (consult the **ENVIRONMENT** and the **FILE FILTERS** sections below).

fz [*on, off*]

Toggle full directory size (long view mode only).

hf, hh, hidden [*on, off, status*]

toggle hidden files.

history [*edit [APP]*] [*clear*] [*-n*] [*on, off, status, show-time*]

with no arguments, it prints the commands history list (use *show-time* to print timestamps as well). 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 as required (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 also use the *HistIgnore* option in the configuration file to prevent specific command lines from being added to the history list).

You can use the exclamation mark (!) to perform some history commands:

!<TAB>: List history entries

!!: 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 the command starting with *STRING*. Tab completion is available in this case: *!STRING<TAB>*.

icons [*on, off*]

toggle icons.

Note: Depending on how the terminal renders icons, the apparent space between icons and file-names may not be the most appropriate. This space can be adjusted using the *IconsGap* option in the configuration file (valid values: 0, 1, 2).

j [*--purge [NUM]*] [*--edit*], jc, jl, jp [*STR*]..., **je**

j is the fastest way of using **Kangaroo**, a **directory jumper** function used 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 **(1)(2)**, printing: **a)** order number of the corresponding entry, **b)** total sum of visits, **c)** days since the first visit, **d)** hours since the last visit, **e)** the rank value, and **f)** the directory name itself. An asterisk next to the rank value means that the corresponding directory will not be removed from the database, despite its rank, either because it has been visited in the last 24 hours, or because it is bookmarked, pinned, or currently active in some workspace.

(1) To prevent a directory from being *added* to the jump database use the *DirhistIgnore* option in the main configuration file.

(2) To prevent a directory from being *removed* from the jump database, edit the database (*j edit*) and prepend a plus sign (+) to the corresponding line.

Otherwise, if a query string is provided as parameter, *j* searches for this string in the database and

cd to the best ranked matching entry. Example: *j Down* 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 can 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 to it without performing any query.

The backslash (\) and the slash (/) can 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 respectively. 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, 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 four modifiers: 'e', 'p', 'c', and 'l', the first standing for "edit", the second for "parent", the third for "child", and the last one for "list". Thus, *je* (or *j --edit*) will open the jump database to be edited as required; *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. Finally, *jl* just prints the matches for the given query string(s), but without changing the current directory. Examples:

Command	Description
<i>jp foo</i>	Change to the best ranked parent directory containing the string "foo".
<i>jc bar test</i>	Change to the best ranked child directory containing the string "bar" and "test"
<i>jl foo</i>	Print all entries in the database containing the word "foo"

Use the *--purge* option to shrink the database. Without further parameters, *--purge* removes all non-existent (un-stat'able) directories from the database. If a numeric parameter is passed, by contrast, all entries ranked below this number will be removed from the database. For example, *j --purge 100* will remove all entries ranked below 100.

You can also manually edit the database file using the *je* (or *j --edit*) command: edit whatever needs to be edited, save changes, and close the editor. This is useful, for example, to remove a specific entry/directory from the database (however, bear in mind that if the directory is in the directory history, it will not be removed from the jump database).

To mark an entry as permanent (prevent it from being removed from the database), follow any of these procedures:

- Bookmark it.
- Edit the jump database (*je* or *j --edit*) and prepend a plus sign (+) to the corresponding entry.

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. Enter ++ to perform a fuzzy search over the jump database.

Take a look at the *dh* command as well.

- k** if running in long view, toggle follow-links (*Alt++* is also available). See the *-L, --follow-symlinks-long* command line switch.
- kk** toggle max-filename-len (*Ctrl+Alt+l* is also available)
- kb, keybinds** [*list*] [*bind FUNCTION*] [*edit [APP]*] [*conflict*] [*reset*] [*readline*]
with no argument (or if the argument is *list*), prints the current keybindings and their associated functions.

To change a keybinding use the *bind* subcommand.

Type *kb bind <TAB>* to get the list of bindable functions.

Enter *kb bind FUNCTION* to set a new keybinding for FUNCTION. For example, to bind the function *previous-dir* to a new key, enter *kb bind previous-dir*. You'll see a little prompt: press the key combination you want to associate to the specified function and the press **Enter** (while in this prompt, press **Ctrl+d** to abort or **Ctrl+c** to clear the current line).

To manually edit your keybindings use the *edit* option (the keybindings file will be opened with APP, if specified, or with the default associated application otherwise).

If you somehow messed up your keybindings, you can check for keybinding conflicts with the *conflict* option, or use the *reset* option to create a fresh keybindings file with the default values.

To unbind a function run *kb edit* and comment out the corresponding entry. Note that some functions may have several entries, associating them to multiple keybindings: comment them out all if required.

To list readline keybindings (defined in *~/.config/clifm/readline.clifm*), use the *readline* option. The syntax is the same as the one used by readline's *.inputrc* file (consult <http://tiswww.case.edu/php/chet/readline/readline.html#SEC9> for more information.)

- l, le** Create (*l*) or edit (*le*) symbolic links.

The syntax for the *l* command is: *l TARGET [LINK_NAME]*. Note that if *LINK_NAME* is omitted, the symbolic link is created as *TARGET_BASENAME.link* in the current directory.

By default, the link target is created literally (like *ln -s* would). The link creation mode can be set via the *LinkCreationMode* option in the configuration file. Available modes are: *absolute*, *literal*, and *relative* (like *ln -rs* would).

To edit the target of a symbolic link use the *le* command followed by the desired link name. The user will be prompted to enter a new link target, using the current one as template.

- ll, lv** [*on, off*]

Toggle long/detail view mode.

- lm** [*on, off*]

Toggle the light mode. This option, aimed at making file listing faster than the default mode, is especially useful for really old hardware or when working on remote machines (for more information see the **NOTE ON SPEED** section below).

log [*cmd* [*list*, *on*, *off*, *status*, *clear*]] [*msg* [*list*, *on*, *off*, *status*, *clear*]]

Enable, disable, clear, list or check the status of the program logs, either message (errors and warnings) or command logs. Example: *log cmd on*, to enable command logs, or *log msg clear*, to clear/remove message logs.

Consult the **FILES** section below for information about how logs are written to the logs file.

media

NOTE: This command is Linux-specific

List available storage devices and mount/unmount the selected one using either *udev* or *udisks2* (at least one of these must be installed. *udev* 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 (*udev* or *udisks2*), **clifm** will automatically cd to the corresponding mountpoint whenever the mount operation was successful.

When unmounting, and if the current directory is inside the mountpoint, **clifm** will attempt to cd to the previous visited directory, and, if none, to 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** [*open FILE*] [*info FILE*] [*edit [APP]*] [*import*]

This is *Lira*, **clifm**'s file opener.

Use the *open* subcommand to open a file with the default associated application. Note that, since *open* is the default action, it can be omitted. For example: *mm file.pdf*. The same can be achieved more easily via the *open* command: *open file.pdf* (or using the short command, *o file.pdf*). Or, even shorter, just *file.pdf*.

The *info* option prints MIME information about *FILE*: its MIME type, and, if any, the application associated to this filename 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 **FILE OPENER** section below for information about the mimelist file syntax.

Finally, via the *import* option **clifm** will try to import MIME associations 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 association is successfully imported, it will be stored as *mimelist.clifm.XXXXXX* (where XXXXXX is a random six digits alphanumerical string). You can add these new associations to your mimelist file using the *mime edit* command.

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*[@*TEMPLATE*]]... [*DIR*/]...

create new regular files and/or directories.

If a filename ends with a slash (/), it will be taken as a directory name. Else, it will be created as a regular file. E.g.: *n myfile mydir/*, to create a file named *myfile* and a directory named *mydir*. If no filename is provided, the user will be prompted to enter one.

Automatic templates

New regular files will be created from a **template file** if:

1. The file to be created has a filename extension (e.g., *file.html*).
2. A file named like this extension, here *html*, exists in the templates directory **(1)**.

If both conditions are met, running *n file.html* will create a new file named *file.html* which is a copy of the *html* file in the templates directory.

Note that template names are not limited to actual file extensions: you can name your templates whatever you like (with any content you want) provided new files are created using the template name as extension. E.g.: *n file.my_super_cool_template*.

Explicit templates

If a filename is followed by the expression @*TEMPLATE*, where *TEMPLATE* is any regular file found in the templates directory**(1)**, the file will be created as a copy of the corresponding file template. E.g., *n file.sh@my_script.sh*.

Tab completion is available for explicit templates: *n file@<TAB>*

(1) The templates directory is **\$CLIFM_TEMPLATES_DIR**, **\$XDG_TEMPLATES_DIR**, or *~Templates*, in this precedence order.

Filename validation is performed over names before creation. In case of an unsafe name, the user is warned and asked for confirmation.

A name (namely, any component of a path) is considered unsafe if:

1. Starts with a dash (-): command option flags collision
2. Is a reserved keyword/expression (internal): fastback (...), ELN/range (12, 1–45), and MIME/file type expansion (@query, =x)
3. Is a reserved system/shell keyword (~, ', ..)
4. Contains embedded control characters (0x00–0x1f in the ASCII table)
5. Contains embedded shell meta-characters (*?:[]"<>|(){ }&'!~;\$)
6. It is too long (larger than NAME_MAX, usually 255 bytes)

For more information about unsafe filenames consult <https://dwheeler.com/essays/fixing-unix-linux-filenames.html>.

net [*NAME*] [*list*] [*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.clifm*). 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* can be used as a placeholder for *Mountpoint*. *%m* will be replaced by the value of *Mountpoint*.

1.a. Mounting remote filesystems

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 filesystem (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 filesystems

Though originally intended to manage remote filesystems, *net* can also manage **local filesystems**. 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 filesystem, and another with an ISO9660 filesystem:

```
[Sandisk USB]
Comment=Sandisk USB drive
```



```

Mountpoint="/media/usb"
MountCmd=sudo mount -o gid=1000,mask=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*, *mask*, 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 (or via the *list* subcommand), *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 file 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 mountpoint specified in the configuration file and automatically *cd* to the corresponding mountpoint. Example: *net m smb_work*. Since *mount* is the default action, it can be omitted: *net smb_work*.

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. Set up your remotes (*cr --edit*) and then send the file you want (*cr FILE*).

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 builtin file opener (see the *mime* command above). However, if you want to open a file with a different application, add the application name as second argument, e.g. *o 12 leafpad* or *o 12 leafpad*.

If you want to run the program in the background, simply add the ampersand character, as usual: *o 12 &*, *o 12&*, *o 12&* 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.

oc *ELN/FILE...*

Interactively change files ownership

A new prompt is displayed using user and primary group common to all files passed as parameters as ownership template.

Ownership (both user and primary group, if specified) is changed for all files passed as parameters. If the file is a symbolic link, the operation is performed on the target file, and not on the symbolic link itself. Bear in mind that recursion is not supported: use **chown**(1) (with the *-R* option) instead.

Both names and ID numbers are allowed (Tab completion for names is available).

If only a name/number is entered, it is taken as the user who owns the file(s).

Use the *pc* command to edit files permissions.

opener [*default*] [*APPLICATION*]

with no argument, prints the currently used file opener (by default, *Lira*, **clifm**'s builtin opener). Otherwise, set *APPLICATION* (say *rifle* or *xdg-open*) as opener or, if *default* is passed instead, use *Lira*.

ow *ELN/FILE* [*APPLICATION*]

If *APPLICATION* is specified, open *ELN/FILE* with *APPLICATION*. In case you need to add parameters to *APPLICATION*, it is recommended to quote the expression: *ow FILE "APP ARG..."*.

If no *APPLICATION* is specified, the list of available applications associated to *ELN/FILE* (either via its MIME type or its file extension) is printed, allowing the user to choose one of these applications, and then open the file with the selected application.

This command supports tab completion. Type "*ow filename <TAB>*" and those applications able to open *ELN/FILE* will be listed.

p, pp, prop *ELN/FILE...*

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). On the other side, and unlike *p*, *pp* provides information about the dereferenced symlinks (namely, the symlink target) instead of the symlink itself. However, note that, in case of symbolic links to directories, *p* provides information about the link *target* if the provided filename ends with a slash. Otherwise, information about the link *itself* is displayed.

The time format used to display time information can be customized via the *PTimeStyle* option in the configuration file (defaults to "%Y-%m-%d %H:%M:%S.%N %z", where %N stands for nano-second precision).

If you need to list the properties of all files in the current directory, try the long view mode (*ll* or *Alt+l*). Fields displayed in this mode can be customized using the *PropFields* option in the configuration file. For custom timestamp formats use the *TimeStyle* option.

For more information about file details consult the *file-details* help topic: *help file-details*.

pc *ELN/FILE...*

Interactively change files permissions (only traditional Unix permissions are supported).

A new prompt is displayed using actual permissions (in symbolic notation) of the file to be edited as template. If editing multiple files with *different sets of permissions*, only shared permission bits are set in the permissions template.

Bear in mind that, if editing multiple files at once, say *pc sel* or *pc *.c*, the new permissions set will be applied to *all* of them.

Both symbolic and octal notation for the new permissions set are allowed.

Recursively setting file permissions is not supported. Use **chmod**(1) with the *-R* flag instead.

If you just need to toggle the executable permission bit on a file, you can use the *te* command.

Use the *oc* command to edit files ownership.

pf, profile [*ls, list*] [*set, add, del PROFILE*] [*rename PROFILE NEW_NAME*]

with no arguments, prints the name of the currently used profile. Use the *ls* or *list* option to list available profiles. To switch, add, delete, or rename a profile, use the *set*, *add*, *del*, and *rename* options respectively.

pg, pager [*on, off, once, status, [NUM]*]

run or set **Mas, Clifm**'s builtin files pager.

With no parameter, just run the pager (*Alt+O* is also available).

If set to *on*, run the pager whenever the list of files does not fit on the screen.

Set it to any positive integer greater than 1 to run the pager whenever the amount of files in the current directory is greater than or equal to this value, say 1000 (0 amounts to *off* and 1 to *on*).

Set to *once* to run the pager only once (overwriting whatever was its previous value).

While paging, the following keys are available:

?, h: Help

Down arrow, Enter, Space: Advance one line

Page down: Advance one page

q: Stop paging (without printing remaining files)

c: Stop paging (printing remaining files)

Note: To scroll lines up, use whatever your terminal emulator has to offer (e.g.: mouse scrolling or some keybinding).

By default, the pager lists files using the current listing mode (long or short). Use *PagerView* in the configuration file (or *--pager-view* in the command line) to force the use of a specific mode. Possibles values:

auto: Use the current listing mode (default)

long: List files in long view

short: List files in short view

pin [*FILE/DIR*]

pin a file or 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 filename 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.

prompt [*set NAME*] [*list*] [*edit [APP]*] [*unset*] [*reload*]

Manage **clifm**'s prompts. Use the *set* subcommand to temporarily change the current prompt to the prompt named NAME (use the *unset* subcommand to unset the current prompt and set the default one). Available prompts (which can be listed via *prompt list* or *prompt set <TAB>*) are defined in the prompts file (*\$HOME/.config/clifm/prompts.clifm*). To permanently set a prompt, edit your color scheme file (via the *cs edit* command) and set *Prompt* to either a prompt code or a prompt name (as defined in the prompts file).

q, quit, exit

Quit **clifm**.

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 this 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 writes all filenames 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...* [*[/]PATTERN*] [*[-filetype]*] [*[:PATH]*]

mark one or multiple files (either regular files or directories) as selected (send to the Selection Box). *sel* accepts individual elements, range of elements, say 1–6, filenames and paths, just as wildcards (globbing) and regular expressions. Example: *s 1 4–10 ^r file* filename /path/to/file-name*.

If not in light mode, once a file is selected, and if the file is in the current directory, the corresponding filename will be highlighted with a mark (colored according to the value of *li* in the color scheme file (by default bold green)) at the left of the filename (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$ -f :/etc`, or using wildcards: `s *.conf -f :/etc`. Of course, you can also do just `s -f /etc/*.conf`.

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

Glob and regular expressions can 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 *Alt+a* is also available to perform the same operation.

The Selection Box is accessible from 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 *ShareSelbox* is enabled, selected files are stored in `/tmp/clifm/username/.selbox.clifm`. Otherwise, `/tmp/clifm/username/.selbox_profile-name.clifm` is used (this is the default).

Operating on selected files

To operate on one or more selected files use the *sel* keyword (*s:* can be used as well). For example, to print the file properties of all selected files: `p sel` (or `p s:`). Use `s:<TAB>` to list selected files (multi-selection is available if running in *zfz* mode).

Listing selected files

To list selected files use the *sb* command (standing for Selection Box). You can also type `s:<TAB>`.

Deselecting files

To deselect files use the *ds* command. See above. You can also press *Alt+d* to deselect all files at once.

Note: If there is a file named *sel* in the current directory, use `./sel` to distinguish it from the *sel* keyword. For example, enter `p ./sel` to tell **clifm** that you want to get the properties of the file named *sel* rather than the properties of the currently selected files.

For more information 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 order. Else, sort files by *METHOD*, where *METHOD* is one of: 0=none, 1=name, 2=size, 3=atime, 4=btime, 5=ctime, 6=mtime, 7=version, 8=extension, 9=inode, 10=owner, 11=group, 12=blocks, 13=links, or 14=type (e.g.: *st atime* or *st 3*). Methods 10 and 11 sort by owner and group ID names if using ID names in long view (see the *PropFields* option in the configuration file). Else, ID numbers are used. The default order is *version*.

By default, files are sorted from less to more (e.g.: from 'a' to 'z' if sorting by **name**). Use the *rev* subcommand to invert this order. E.g.: *st rev* or *st inode rev*. Switch back to the previous state running *st rev* again.

Take a look at the configuration file for extra sorting options (*ListDirsFirst*, *PrioritySortChar*, *ShowHiddenFiles*).

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

t, trash [*ELN/FILE*]... [*ls, list*] [*clear, empty*] [*del [FILE]...*]

move specified files to the trash can (e.g. *t file1 file2*).

With no argument (or by passing the *ls* option), it prints the list of currently trashed files. The *clear* (or *empty*) sucommand removes all files from the trash can, while the *del* subcommand lists trashed files allowing the user to permanently remove one or more trashed files. If using *del*, tab completion to list/select currently trashed files is available.

The trash directory is *\$XDG_DATA_HOME/Trash*, falling back to *\$HOME/.local/share/Trash*. To set an alternative trash directory use the *-T, --trash-dir* command line option.

Since this trash system follows the Freedesktop specification, it is able to handle files trashed by different Trash implementations.

To restore trashed files (to their original location) see the *undel* command below.

tag [*add, del, list, list-full, new, merge, rename, untag*] [*FILE*]... [[:]*TAG*]

tag is the main *Etiqueta* command, **clifm**'s builtin files tagging system. See the **FILE TAGS** section for a complete description of this command.

te *FILE*...

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*]...

If filenames 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 the user to choose one or more of these files to be restored. Use the ***, *a* or *all* parameters to restore all trashed files at once. Tab completion to list/select currently trashed files is available.

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.

vv *FILE*... *DIR*

copy *FILE*(s) to *DIR* and bulk rename them at once.

ver, version

show **clifm** version details.

view [*edit* [*APP*] | *purge*]

preview files in the current directory (full screen). Requires **fzf**(1). *Alt+-* is also available.

By pressing Enter or Right, the currently highlighted file will be selected and *view* closed. To select multiple files, mark them with the TAB key and then press Enter or Right to confirm. To quit *view* press Escape or the Left arrow key.

Run *view purge* to purge the thumbnails directory (`$XDG_CACHE_HOME/clifm/thumbnails`) of dangling thumbnails.

To edit the previewer configuration file enter *view edit*, or *view edit vi* to open it with a specific application, in this case, **vi**(1).

For **image previews** consult the Wiki (<https://github.com/leo-arch/clifm/tree/master/misc/tools/imgprev>) or enter *help image-previews*.

For further information consult the **SHOTGUN** section below.

ws [*NUM*/*NAME* [*unset*], +, -]

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 the workspace number *NUM*, *NAME* to switch to the workspace named *NAME*, the plus sign (+) to switch to the next workspace, and the minus sign (-) to switch to the previous workspace.

To unset a workspace use the *unset* subcommand preceded by the workspace (either number or name) to be unset. For example: *ws 2 unset*.

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 current working directory.

Though by default workspaces are unnamed, you can name them however you like using the *WorkspaceNames* option in the configuration file.

Use autocommands to persistently set options per workspace, for example, to always list files in the third workspace in long view. See the **AUTOCOMMANDS** section below for more information.

Make local settings private to the current workspace by setting the *PrivateWorkspaceSettings* option to *true* in the configuration file: settings changed via either the command line or keyboard shortcuts (say Alt+l, to toggle the long view) will apply only to the current workspace and will be remembered even when switching workspaces.

To directly operate on a workspace (namely, the path it points to) you can use the *w:* construct followed by a workspace number or name. For example, to copy all .png files in the current directory to the third workspace, enter *c *.png w:3*. Press TAB immediately after *w:* to get the list of available workspaces.

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.

Shell-builtins implementations

pwd [*-LP*]

print the current working directory

export *NAME=VALUE...*

export variables to the environment

umask [*VALUE*]

print/set the current umask value

unset *NAME*

remove a variable from the environment

5. FILE FILTERS

Clifm provides multiple ways to filter the current list of files:

a) Hidden files: via the **-A** and **-a** command line flags, the *hh* command, and the *Alt+*. keybinding.

Files listed in a file named *.hidden* in the current directory will be hidden as well whenever hidden files are not shown (wildcards are supported).

b) Directories: via the **--only-dirs** command line switch and the *Alt+*, keybinding.

c) Filenames and file types: either via a regular expression or a file type character (see below) using the *ft* command (the *Filter* option in the configuration file and the **CLIFM_FILTER** environment variable are also available). For example, to exclude backup files (ending with a tilde):

CLIFM_FILTER='!.*~\$' clifm

or (in the configuration file):

Filter='!.*~\$'

or (via the *ft* command):

ft !.*~\$

See the *ft* command for a few more examples.

d) Filtering files via the TAB key:

You can filter files **by name** using wildcards. For example: *p *.mp3<TAB>* (or */*.mp3<TAB>*) to get a list of MP3 files in the current directory.

Files can also be filtered **by MIME-type** via the *'@'* keyword. Type *@<TAB>* to list all MIME-types found in the current directory, or *@query<TAB>* to list all files whose MIME-type includes the string "query". For example, *@image<TAB>* will list all files in the current directory whose MIME type includes the string "image".

Finally, files can be filtered as well **by file type** using the '=' keyword followed by a file type character (see below). For example, =l<TAB> to get a list of symbolic links in the current directory.

Note: If using tab completion in fzf mode, multi-selection is allowed (except in the case of @<TAB>).

Available file type characters:

- b**: Block devices
- c**: Character devices
- C**: Files with capabilities (1)(2)
- d**: Directories
- D**: Empty directories
- f**: Regular files
- F**: Empty regular files
- g**: SGID files (2)
- h**: Multi-hardlink files (directories excluded)
- l**: Symbolic links
- L**: Broken symbolic links
- o**: Other-writable files (2)
- p**: FIFO/pipes (2)
- s**: Sockets (2)
- O**: Doors (Solaris only)
- P**: Event ports (Solaris only)
- t**: Files with the sticky bit set (2)
- u**: SUID files (2)
- x**: Executable files (2)

(1) Only for tab completion

(2) Not available in light mode

e) Grouping files (via automatic expansion):

By means of the above features, you can easily group and operate on groups of files. For example, this command:

```
vt b: @image =x sel t:work *.txt
```

opens a virtual directory (see the **VIRTUAL DIRECTORIES** section below) automatically expanding the above expressions as follows:

Expression	Description
b:	All your bookmarks (paths)
@image	All image files (CWD)
=x	All executable files (CWD)
sel	All selected files
t:work	All files tagged as <i>work</i>
*.txt	All .txt files (CWD)

6. KEYBOARD SHORTCUTS

The following is the list of default keyboard shortcuts:

Key	Description
Ctrl+Alt+j	Toggle the vi editing mode
Right, Ctrl+f	Accept the current suggestion
Alt+Right, Alt+f	Accept the first suggested word (up to the first slash or space)
Alt+c	Clear the current command line buffer
Alt+q	Delete last word (up to last slash or space)
Alt+i, Alt+.	Toggle hidden-files
Alt+l	Toggle long-view
Alt++	Toggle follow-links (long view only)
Alt+g	Toggle list-directories-first
Alt+,	Toggle list-only-directories
Ctrl+Alt+l	Toggle max-filename-length
Ctrl+Alt+i, Alt+Tab	Toggle disk-usage-analyzer
Alt+w	Toggle full-path-filenames (virtual directories)
Ctrl+l	Refresh the screen (reprint the list of files in the current directory)
Alt+t	Clear program messages
Alt+m	List mountpoints
Alt+b	Launch the Bookmark Manager
Alt+h	Show the directory history
Alt+n	Create new file or directory
Alt+s	Open the Selection Box
Alt+-	Launch the file previewer (<i>view</i> command)
Alt+a	Select all files in the current directory
Alt+d	Deselect all files
Alt+0	Run MAS, the file pager
Alt+p	Change to the pinned directory
Alt+1	Switch to workspace 1
Alt+2	Switch to workspace 2
Alt+3	Switch to workspace 3
Alt+4	Switch to workspace 4

Key	Description
Alt+r	Change to the root directory
Alt+e, Home	Change to the home directory
Alt+u, Shift+Up	Change to the parent directory
Alt+j, Shift+Left	Change to the previously visited directory
Alt+k, Shift+Right	Change to the next visited directory
Ctrl+Alt+o	Switch to the previous profile
Ctrl+Alt+p	Switch to the next profile
Ctrl+Alt+a	Archive selected files
Ctrl+Alt+e	Export selected files
Ctrl+Alt+r	Rename selected files
Ctrl+Alt+d	Remove selected files
Ctrl+Alt+t	Trash selected files
Ctrl+Alt+v	Copy selected files to the current directory
Alt+y	Toggle light-mode
Alt+z	Switch to previous sorting method
Alt+x	Switch to next sorting method
Ctrl+Alt+x	Launch a new instance of clifm
Ctrl+y	Copy the contents of the line buffer to the clipboard (1)
F1	Go to the manpage
F2	List commands
F3	List keybindings
F6	Open the MIME list file
F7	Open the shotgun configuration file
F8	Open the current color scheme file
F9	Open the keybindings file
F10	Open the main configuration file
F11	Open the bookmarks file
F12	Quit

(1) This shortcut is bound to the *xclip* plugin. See the **PLUGINS** section below for more information.

Customizing keybindings

The above are the default keyboard shortcuts. However, they can be customized using the *kb bind* command (for more information consult the description for the *kb* command above).

The keybindings configuration file can also be manually edited using the *kb edit* command (for more details take a look at the description provided by this file itself).

Readline keybindings

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 these keybindings by editing the `~/.config/clifm/readline.clifm` file, following the same rules used by readline itself for the `~/.inputrc` file. For more information consult the readline documentation (**readline(3)**).

Keybindings for plugins

clifm provides sixteen 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 plugin x (where ‘ x ’ is the plugin number [1–16]) to your plugin using the *actions edit* command. E.g.: "plugin1=myplugin.sh"
3. Set a keybinding for plugin x using the *kb edit* command. E.g.: "plugin1:\M–7"

Kitty keyboard protocol support

The Kitty Keyboard Protocol offers a significant improvement over the current handling of keyboard events in terminals. This protocol utilizes *CSI u* escape sequences and introduces several enhancements, including the ability to use extra modifier keys such as Super, Meta, and Hyper.

To enable the Kitty Keyboard Protocol, follow these steps:

1. Download the specially crafted keybindings configuration file designed to handle *CSI u* escape sequences (<https://github.com/leo-arch/clifm/blob/master/misc/kitty/keybindings.clifm>).
2. Configure **clifm** to use this new file instead of the default one (`~/.config/clifm/keybindings.clifm`) by using the `–k` command line switch. Alternatively, you can replace the default file with the new one.
3. If your terminal is not already set up to send *CSI u* sequences, use the `–kitty–keys` command line switch.

Summarizing, and once you have the replacement keybindings file in place, run the following command:

```
clifm -k /path/to/keybindings/file --kitty-keys
```

Important

When using the Kitty Keyboard Protocol, the Ctrl+d shortcut (used to quit secondary prompts) will no longer work. To resolve this issue, add the following line to your *kitty.conf* file:

```
map control+d send_text kitty \x04
```

This will force the terminal to emit the old value for this specific key binding, providing a workaround for the issue.

Troubleshooting

Some of these default keybindings may 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, set your custom keybindings.

Kernel builtin 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 them 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 encoding. 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.

7. THEMING

All customization settings (theming) are made from a single configuration file (the color scheme file), installed by default in `$XDG_DATA_DIRS/clifm/colors` (usually `/usr/local/share/clifm/colors` or `/usr/share/clifm/colors`), though color scheme files found in `$XDG_CONFIG_HOME/clifm/colors` (usually `$HOME/.config/clifm/colors`) take precedence.

Note: Color scheme files are copied automatically to the local colors directory when running the `cs edit` command.

Each color scheme may include any (or all) of the below options:

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 ELNs, file properties bits, suggestions, syntax highlighting, etc. See the **COLORS** section below.

ExtColors = Colors for files based on filename extensions. See the **COLORS** section below.

DateShades = A comma delimited list of colors used to print timestamps (long view). Consult the default color scheme file for more information.

SizeShades = A comma delimited list of colors used to print file sizes (long view). Consult the default color scheme file for more information.

DirIconColor = Color for the directory icon (when icons are enabled). See the **COLORS** section below. Only when using icons-in-terminal or Nerfonts. If using rather emoji-icons (default build), this option is ignored.

Prompt = Define Clifm's prompt. 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`. Enter just `cs` to list available color schemes (tab completion is available). To edit the current color scheme enter `cs edit`.

1. COLORS

If 256 colors support is detected for the current terminal, and not set in any other way (either via the *ColorScheme* option in the configuration file or the `--color-scheme` command line switch), **clifm** will attempt to load the 256 colors version of the default color scheme: default-256. Otherwise, it falls back to the 16

colors version.

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

a. 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: *name=value*, where *name* refers to an interface element, and *value* to the color to be used by this element.

This is the list of **file type codes** (you will find them in the *FiletypeColors* section of the current color scheme file):

- di = directory
- ed = empty directory
- nd = directory with no read/exec permission (**1**)
- fi = regular file
- ef = empty regular file
- nf = file with no read permission (**1**)
- 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
- oo = door/port (Solaris only)
- no = unknown file type
- uf = unaccessible files (**fstatat**(3) error)

(**1**) If unset, the corresponding file type color is used and an exclamation mark is printed before the filename in the file list (provided icons are disabled -otherwise the lock icon is used- and **clifm** is not running in light mode -in light mode access checks are not performed). The color used for the exclamation mark is *xf* (see below).

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

Suggestions

- sb = shell builtins
- sc = aliases and shell command names
- sd = internal commands description
- sf = ELNs, bookmarks, tag, and filenames
- sh = commands history entries
- sx = suggestions for **clifm**'s internal commands and parameters
- sp = suggestions pointer (e.g.: `56 > filename`, where `'>'` is the suggestion pointer)
- sz = filenames (fuzzy)

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 `'$'`)
 hw = Backslash (aka whack)

Prompt elements

li = selected files
 ti = trash indicator
 ac = autocommand indicator
 em = error message indicator
 wm = warning message indicator
 nm = notice message indicator
 ro = read-only mode indicator
 si = stealth mode indicator
 tx = command line text (regular prompt)

File properties

db = file allocated blocks
 dd = last access/change/modification time(**1**)
 de = file inode number (long view only)
 dg = group ID (provided the user has access to the file)
 dk = number of links (long view only)
 dn = dash (unset property)
 do = octal value for file properties
 dp = SUID, SGID, sticky bit
 dr = read permission bit
 dt = timestamp identification mark(**2**)
 du = user ID (provided the user has access to the file)
 dw = write permission bit
 dxd = executable permission bit (directories)
 dxr = executable permission bit (regular files)
 dz = size(**1**)

(**1**) If unset (default), gradient colors are used (based on file size and file age).

(**2**) If unset (default), a dimmed version of the current timestamp color is used.

NOTE: For a better graphical representation of file properties, 256 colors are used if possible (otherwise, **clifm** falls back to 16 colors).

Miscellaneous interface elements

fc = files counter
 df = default color
 dl = dividing line
 el = ELN color
 lc = symbolic link indicator (*ColorLinksAsTarget* only)
 mi = misc indicators (disk usage, sort method, bulk rename, jump database list)

ts = matching suffix for possible tab completed entries
 tt = tilde for truncated filenames
 wc = welcome message
 wsN = color for workspace N (1–8)
 xs = exit code: success
 xf = exit code: failure

b. Supported colors

4-bit, 8-bit (256-colors), and 24-bit (true colors) colors are supported.

Colors are defined basically as SGR sequences (excluding the initial escape character and the ending 'm'), the same sequences used by the **LS_COLORS** environment variable. However, shortcuts for 8-bit (256-colors) and 24-bit (true color) colors are available. For example:

```
31      4-bit
38;5;160 8-bit
@160    8-bit (short)
38;2;255;0 24-bit
#ff0000 24-bit (short, HEX)(1)
```

(1) Both three and six digits hexadecimal colors (lower or uppercase) are supported. For example, **#f00** amounts to **#ff0000**.

A *single* attribute can be added to hex colors and 256 colors (in the form @NUM) using a dash and an attribute number (#RRGGBB-[1-9] or @NUM-[1-9]), where 1-9 is:

- 1: Bold or increased intensity
- 2: Faint, decreased intensity or dim
- 3: Italic (Not widely supported)
- 4: Underline
- 5: Slow blink
- 6: Rapid blink
- 7: Reverse video or invert
- 8: Conceal or hide (Not widely supported)
- 9: Crossed-out or strike

Note: Some attributes may not be supported by all terminal emulators.

For example, for bold red the hex code is **#ff0000-1**, while the 256-colors code is **@160-1**. (for more information about SGR sequences consult https://en.wikipedia.org/wiki/ANSI_escape_code).

c. Color names

Xterm-like color names are also supported. For example: **ex=DodgerBlue2**.

This is the list of color names (as defined by **vifm(1)**):

0 Black	86 Aquamarine1	172 Orange3
1 Red	87 DarkSlateGray2	173 LightSalmon3_2
2 Green	88 DarkRed_2	174 LightPink3
3 Yellow	89 DeepPink4_2	175 Pink3
4 Blue	90 DarkMagenta	176 Plum3
5 Magenta	91 DarkMagenta_2	177 Violet
6 Cyan	92 DarkViolet	178 Gold3_2
7 White	93 Purple	179 LightGoldenrod3
8 LightBlack	94 Orange4_2	180 Tan
9 LightRed	95 LightPink4	181 MistyRose3
10 LightGreen	96 Plum4	182 Thistle3
11 LightYellow	97 MediumPurple3	183 Plum2

12 LightBlue	98 MediumPurple3_2	184 Yellow3_2
13 LightMagenta	99 SlateBlue1	185 Khaki3
14 LightCyan	100 Yellow4	186 LightGoldenrod2
15 LightWhite	101 Wheat4	187 LightYellow3
16 Grey0	102 Grey53	188 Grey84
17 NavyBlue	103 LightSlateGrey	189 LightSteelBlue1
18 DarkBlue	104 MediumPurple	190 Yellow2
19 Blue3	105 LightSlateBlue	191 DarkOliveGreen1
20 Blue3_2	106 Yellow4_2	192 DarkOliveGreen1_2
21 Blue1	107 DarkOliveGreen3	193 DarkSeaGreen1_2
22 DarkGreen	108 DarkSeaGreen	194 Honeydew2
23 DeepSkyBlue4	109 LightSkyBlue3	195 LightCyan1
24 DeepSkyBlue4_2	110 LightSkyBlue3_2	196 Red1
25 DeepSkyBlue4_3	111 SkyBlue2	197 DeepPink2
26 DodgerBlue3	112 Chartreuse2_2	198 DeepPink1
27 DodgerBlue2	113 DarkOliveGreen3_2	199 DeepPink1_2
28 Green4	114 PaleGreen3_2	200 Magenta2_2
29 SpringGreen4	115 DarkSeaGreen3	201 Magenta1
30 Turquoise4	116 DarkSlateGray3	202 OrangeRed1
31 DeepSkyBlue3	117 SkyBlue1	203 IndianRed1
32 DeepSkyBlue3_2	118 Chartreuse1	204 IndianRed1_2
33 DodgerBlue1	119 LightGreen_2	205 HotPink
34 Green3	120 LightGreen_3	206 HotPink_2
35 SpringGreen3	121 PaleGreen1	207 MediumOrchid1_2
36 DarkCyan	122 Aquamarine1_2	208 DarkOrange
37 LightSeaGreen	123 DarkSlateGray1	209 Salmon1
38 DeepSkyBlue2	124 Red3	210 LightCoral
39 DeepSkyBlue1	125 DeepPink4_3	211 PaleVioletRed1
40 Green3_2	126 MediumVioletRed	212 Orchid2
41 SpringGreen3_2	127 Magenta3	213 Orchid1
42 SpringGreen2	128 DarkViolet_2	214 Orange1
43 Cyan3	129 Purple_2	215 SandyBrown
44 DarkTurquoise	130 DarkOrange3	216 LightSalmon1
45 Turquoise2	131 IndianRed	217 LightPink1
46 Green1	132 HotPink3	218 Pink1
47 SpringGreen2_2	133 MediumOrchid3	219 Plum1
48 SpringGreen1	134 MediumOrchid	220 Gold1
49 MediumSpringGreen	135 MediumPurple2	221 LightGoldenrod2_2
50 Cyan2	136 DarkGoldenrod	222 LightGoldenrod2_3
51 Cyan1	137 LightSalmon3	223 NavajoWhite1
52 DarkRed	138 RosyBrown	224 MistyRose1
53 DeepPink4	139 Grey63	225 Thistle1
54 Purple4	140 MediumPurple2_2	226 Yellow1
55 Purple4_2	141 MediumPurple1	227 LightGoldenrod1
56 Purple3	142 Gold3	228 Khaki1
57 BlueViolet	143 DarkKhaki	229 Wheat1
58 Orange4	144 NavajoWhite3	230 Cornsilk1
59 Grey37	145 Grey69	231 Grey100
60 MediumPurple4	146 LightSteelBlue3	232 Grey3
61 SlateBlue3	147 LightSteelBlue	233 Grey7
62 SlateBlue3_2	148 Yellow3	234 Grey11
63 RoyalBlue1	149 DarkOliveGreen3_3	235 Grey15
64 Chartreuse4	150 DarkSeaGreen3_2	236 Grey19
65 DarkSeaGreen4	151 DarkSeaGreen2	237 Grey23

66 PaleTurquoise4	152 LightCyan3	238 Grey27
67 SteelBlue	153 LightSkyBlue1	239 Grey30
68 SteelBlue3	154 GreenYellow	240 Grey35
69 CornflowerBlue	155 DarkOliveGreen2	241 Grey39
70 Chartreuse3	156 PaleGreen1_2	242 Grey42
71 DarkSeaGreen4_2	157 DarkSeaGreen2_2	243 Grey46
72 CadetBlue	158 DarkSeaGreen1	244 Grey50
73 CadetBlue_2	159 PaleTurquoise1	245 Grey54
74 SkyBlue3	160 Red3_2	246 Grey58
75 SteelBlue1	161 DeepPink3	247 Grey62
76 Chartreuse3_2	162 DeepPink3_2	248 Grey66
77 PaleGreen3	163 Magenta3_2	249 Grey70
78 SeaGreen3	164 Magenta3_3	250 Grey74
79 Aquamarine3	165 Magenta2	251 Grey78
80 MediumTurquoise	166 DarkOrange3_2	252 Grey82
81 SteelBlue1_2	167 IndianRed_2	253 Grey85
82 Chartreuse2	168 HotPink3_2	254 Grey89
83 SeaGreen2	169 HotPink2	255 Grey93
84 SeaGreen1	170 Orchid	
85 SeaGreen1_2	171 MediumOrchid1	

Just as with hex colors, a single attribute can be appended to color names. For example, *SteelBlue1-1* to get the bold version of this color.

d. Color variables

Up to 128 custom color variables can be used via the *define* keyword to make it easier to build and read theme files. Example:

```
define FTYPE_DIR=31
define IFACE_ELN=4;38;2;255;255;0;48;2;0;14;191

FiletypeColors="di=FTPYE_DIR:"
InterfaceColors="el=IFACE_ELN:"
```

These variables can only be used for *FiletypeColors*, *InterfaceColors*, *ExtColors*, and *DirIconColor*. The *Prompt* line (if using a prompt code) uses full SGR sequences or prompt-specific color codes instead.

e. Examples

A few examples to put all this together:

```
fi=4;31 (regular files are 4-bit underlined red)
di=@33-1 (directories are 8-bit bold light-blue)
ln=#5fd7ff (symbolic links are 24-bit light cyan)
so=Yellow3 (socket files are Yellow3)
```

More complex combinations can be achieved using complete SGR sequences (for example, to add a background color). E.g.:

```
fi=4;38;2;245;76;48;2;0;0;255
```

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

NOTE: It may 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** a specific value even if the value of **TERM** says

otherwise. Supported values are: 8, 16, 256, truecolor (or 24bit).

To see a colored list of the currently used colors run the *cs preview* command.

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 is taken from the *Prompt* line in the color scheme file using a prompt name as defined in the prompts file, for example, *Prompt="security-scanner"*.

Each prompt is built 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 `$(cmd)`), prompt modules (in the form `${module}`), color codes (in the form `%{color}`), string literals, and escape sequences can be used to build the prompt.

Consult the prompts file (using the *prompt edit* command) for detailed information and examples on how to build a prompt.

By default, for instance, **clifm**'s prompt line is this:

```
"%{reset}\I[\S%{reset}]]\I \A \u:\H %{cyan}\w%{reset}\n<\z%{reset}> %{blue}\$%{reset} "
```

which once decoded should look something like this:

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

with the workspace number printed in blue, the path in cyan, the last exit status in green (or red in case of error), and the dollar sign in blue.

A more "classic" prompt can be generated as follows:

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

or, using now command substitution:

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

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 'S' indicates that there are elements in the Selection Box. In the same way, a cyan '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 *Notifications* is set to "false" in the prompts file, the above notifications won't be printed by the prompt, but are still available to the user as escape codes (see above) and environment variables (see the **ENVIRONMENT** section below) to build custom prompts.

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.

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

```
Is quoted (e.g.: "string" or 'string')
```

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

The warning prompt can be customized by means of the same rules used by the regular prompt. To use a custom warning prompt, modify the *WarningPrompt* line in the prompts file (via the *prompt edit* command). It defaults to

```
"%{reset}%{b:red}(!)%{n:dim} > "
```

the last line of the regular prompt will become "(!) > ", with "(!)" printed in bold red.

To disable this feature use the `--no-warning-prompt` command line switch or set the *EnableWarningPrompt* option to **false** in the prompts file.

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

3. THE DIVIDING LINE

The line dividing the current list of files and the prompt. It can 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, specify two 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, if 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.

8. 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. *s:* can be used interchangeably (with the difference that *s:* can be used as first word, and not only as parameter to other commands).

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 to some directory: *cp sel 12*, or *mv sel 12* (provided the ELN 12 corresponds to a directory), respectively.

If the destination directory is omitted, selected files are copied to 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 then press `Enter`, as usual.

Tab completion

There are four modes for tab completion: *standard* (interface provided by `readline`), *fzf*, which depends on **fzf** (<https://github.com/junegunn/fzf>) (version 0.18.0 or later), *fnf* (<https://github.com/leo-arch/fnf>), and *smenu* (<https://github.com/p-gen/smenu>). By default, if the *fzf* binary is found in **PATH**, **clifm** will attempt to use *fzf* to display completions. You can force the use of the remaining modes via the `--stdtab`, `--fnftab`, and `--smenutab` command line switches. The *TabCompletionMode* option in the configuration file can be used to permanently set the tab completion mode.

If using the *fzf* mode, the completions interface can 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,border:7:dim --bind tab:accept,right:accept,left:abort,alt-p:toggle-preview --inline-info --layout=reverse-list --preview-window=wrap,border-left
```

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

If set neither in *FzfTabOptions* nor in **FZF_DEFAULT_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**), set *FzfTabOptions* to *none*.

File previews are available in *fzf* mode via *shotgun*. See the **SHOTGUN** section above.

Image previews are available via the *clifmimg* plugin. Run `help image-previews` for more information.

If using the *smenu* mode, the interface can be customized using the **CLIFM_SMENU_OPTIONS** environment variable. For example:

```
export CLIFM_SMENU_OPTIONS="-a t:2,b b:4 c:r ct:2,r sf:6,r st:5,r mt:5,b"
```

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

For information about how to customize *fnf* consult **fnf**(1).

Clifm can perform fuzzy tab completion (just as suggestions) for filenames and paths (e.g. "dwn" is completed/suggested as "Downloads"). To enable this feature use the `--fuzzy-matching` command line switch or set *FuzzyMatching* to *true* in the configuration file.

Besides the default *tab completion* for command **names and paths**, you can also expand **ELNs** 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 filename: "this is a test@version{ 1 }" is expanded as follows: `this\ is\ a\ test\@version\{ 1\}`

ELNs and **ELN ranges** will be also automatically expanded, provided the corresponding **ELNs** actually exist, that is to say, provided some filename 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. Note that the second field of a range can be omitted, in which case the **ELN** of the last listed file is assumed (e.g.: provided there are 100 listed files, `12-` is equivalent to `12-100`).

Since ranges can be a bit tricky, tab completion is available to make sure this range actually includes the desired filenames.

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, users home directory, workspaces, wildcards*, file types*, environment variables, bookmarks, profiles, color schemes, file tags, commands 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 ELNs 2, and 23 to 31, to the current working directory.

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

9. FILE OPENER

As **clifm**'s builtin file 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 filename), 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.clifm*) 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*).

A note about MIME types

File MIME types are determined using *libmagic* –the same library used by **file**(1)–, which, though highly reliable, is not bullet-proof. Sometimes it fails, either because the appropriate MIME type is not in its database, or because the database is just wrong. In either case, you can manually map file extensions to MIME types using a specific file (by default, *~/mime.types*).⁽¹⁾

By way of example, *libmagic* knows nothing about ILBM image files, and returns *application/zip* for OpenRaster images. Create *~/mime.types* with the following content:

```
image/x-ilbm    iff lbm
image/openraster ora
```

Restart **clifm** and these MIME types will be immediately associated to all files having the specified extensions (to test it, run *mm info* on any of these files).⁽²⁾

If required, edit the *mimelist* and the *preview* files (*mm edit* and *view edit*, to specify how files are opened and previewed respectively), and add a new line handling the corresponding MIME types (say, *image/x-ilbm=APP* and *image/openraster=APP*). See below for more information.

⁽¹⁾ To use a different file, set **\$CLIFM_MIMETYPES_FILE** to the desired file, for example, *CLIFM_MIMETYPES_FILE=/etc/mime.types clifm*.

(2) In case of issues, bear in mind that the *mime.types* file is read top to bottom, and that, in case of conflicts (mostly duplicate extensions), only the last entry is effective.

Important: Though sometimes convenient, determining file types by means of filename extensions alone is unreliable: a file, no matter its type, can bear any file extension, without restriction. Because of this, you might end up executing a command that was not intended due to wrong file identification. Be extra careful when doing this.

1. Syntax

In its most basic form, each line in the MIME list file consists of:

a) A left value: this is just a regular expression indicating what we are trying to match (it can be a filename, a file extension, or a MIME type).

b) A right value: a semicolon separated list of commands to be used as the opening application (the first existing program found in this list will be used).

For example:

```
^text/.*=leafpad
```

which is to be read as follows: Open text files (in this case we are partially matching a MIME type) using **leafpad**.

As explained below, this basic rule can be modified to get much more control on **what** we are matching and **how** we execute the opening application.

The syntax is this:

```
[!][X:][N:]REGEX=CMD [ARGS] [%[f,x]] [![E,O]] [&]; ...
```

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

2. The left value (REGEX)

2.1. The X prefix

Without any prefixes, the rule will attempt to match MIME types, disregarding if we are running on a graphical or non-graphical environment. For example,

```
^text/.*=leafpad
```

instructs **lira** to open all text files using **leafpad**, no matter if we are running on a graphical or non-graphical environment.

However, we usually do not want to use **leafpad** if we are not running on a graphical environment. In this case, we can write a double rule as follows:

```
X:^text/.*=leafpad
!X:^text/.*=nano
```

where the first rule (via the **X** prefix) is intended for use on graphical environments, where we can use **leafpad**, and the second one (via the **!X** prefix) for non-graphical environments, where we rather prefer to use **nano**.

2.2. The N prefix

Sometimes MIME types are not enough to identify a file, or we just want to match a specific filename. In this case, we can use the **N** prefix to tell **Lira** that we want to match a filename instead of a MIME type. For example:

```
X:N:^filename.txt$=leafpad
```

in which case we want to match exactly the filename *filename.txt* (no matter its MIME type).

If we want to match file extensions, instead of entire filenames, we can use a regular expression, as follows:

X:N:*.txt\$=leafpad

Here, we are not matching a specific filename, but a specific file extension, so that the rule reads as follows: open all files ending with .txt using **leafpad**.

3. The right value (CMD)

The right value is a semicolon separated list of commands, each of which contains a command, and optionally, command arguments and modifiers. For example:

X:N:*.txt\$=leafpad --sync,geany,mousepad,nano

which means: Open .txt files (graphical environments only) using *leafpad --sync*, or, if not found, *geany*, *mousepad*, or *nano*, in this order. The file to be opened will be appended to the command string, say *leafpad --sync FILE*.

3.1. The %f placeholder

Use the **%f** placeholder to specify the position of the file 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 to be opened will be appended to the command string. Thus, this: *mpv --terminal=no* amounts to this: **mpv --terminal=no FILE**.

3.2. STDERR and STDOUT

Sometimes we might need to silence either standard error (**STDERR**), standard output (**STDOUT**), or both. Use **!E** and **!O** to silence them respectively. Both can be used together: **!EO**. Example: *leafpad %f !EO*, or, to silence only **STDERR**: *leafpad %f !E*.

3.3. Run in the background

The ampersand character (**&**) can be used, as usual, to run the opening application in the background. Example: *leafpad %f &*.

3.4. The %x flag

The **%x** flag is a shorthand for "**%f !EO &**": the command will be executed in the background and both **STDOUT** and **STDERR** will be silenced. As a plus, the command is executed in a new session, i.e. detached from the running terminal (via **setsid**(3)). This flag is recommended to open files via graphical applications. Examples:

For GUI applications:

APP %x

For terminal applications:

TERM -e APP %x

Replace **TERM** and **APP** by the appropriate values (say, *xterm* and *vi* respectively). The **-e** option might vary depending on the terminal emulator used.

Note: In case of archives, the builtin *ad* command can be used as opening application.

3.5. Environment variables

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 associ-

ations:

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

3.6. Using shell scripts

Bear in mind that commands will be executed directly without shell intervention, so that no shell goodies (like pipes, conditions, loops, etc) are available. In case you need something more complex than a single command (including shell capabilities) write your own script and place the path to the script in place of the command. For example:

```
X:text/*:~/scripts/my_cool_script.sh
```

4. Examples:

Match a full filename:

```
X:N:some_filename=nano;vim;vi;emacs
```

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

Open video files with **mpv** in the foreground and silence STDERR:

```
^video/*=mpv %f !E
```

Open video files with **gmplayer** in the background and silence both STDERR and STDOUT:

```
^video/*=gmplayer %f !EO & (or 'gmplayer %x')
```

Match multiple filenames (starting with "str"):

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

Match a single extension:

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

Match multiple extensions:

```
X:N:*.sh|c|py|pl$=geany %x;leafpad %x;nano
```

Match a single mimetype:

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

Match mutiple mimetypes:

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

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 %x
```

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

5. Using a third-party opener

This can be done in two ways:

a. Set *Opener* in the configuration file to the name of the desired opener. For example, to use Ranger's **rifle**(1):

```
Opener=rifle
```

or, if you prefer **xdg-open**(1):

```
Opener=xdg-open
```

b. Tell *Lira* to open all files, no matter the MIME type or filename, via the desired opener. For example:

```
.*=rifle
```

6. Using Clifm as a standalone file opener

Though **clifm** is a file manager, it can be used as a simple file 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.

10. SHOTGUN

1. Tab completion with file previews

Shotgun is **clifm**'s builtin files previewer. Though, as described below, it may be used as a standalone and general purpose file previewer (similar in this regard to **pistol**(1)), it is mainly intended to be used by **clifm**'s tab completion function running in *fzf* mode: every time tab completion is invoked for files, *shotgun* will be executed with the currently highlighted file as argument (as shown below) to generate the preview. Set the *FzfPreview* option in the configuration file to *false* (or run with `--no-fzfpreview`) to disable this feature.

Shotgun is also used by the *view* command to display file previews in full screen.

2. Running as a standalone files previewer

Executed via the `--preview` command line switch, *shotgun* performs file preview for any file passed as argument. For example:

```
clifm --preview myfile.txt
```

This command generates a preview of the file *myfile.txt* and then quits **clifm**.

3. Customization

Previewing applications (based on either MIME type or filename) are defined in a configuration file (`$XDG_CONFIG_HOME/clifm/profiles/PROFILE/preview.clifm`) using the same syntax used by *Lira* (the builtin file opener). See the **FILE OPENER** section above.

You can set an alternative configuration file via the `--shotgun-file` command line switch:

```
clifm --shotgun-file=/path/to/shotgun/config/file --preview=myfile.txt
```

To customize the appearance of the preview window, use the `--preview-window` option in the *FzfTabOptions* line in the current color scheme file. For example, if you want the preview window down the file list (instead of to the right):

```
--preview-window=down
```

Default keybindings for the preview window:

Alt+p: Toggle the preview window on/off
Ctrl+Up / **Shift+Up**: Scroll the preview window up one line
Ctrl+Down / **Shift+Down**: Scroll the preview window down one line
Alt+Up: Scroll the preview window up one page
Alt+Down: Scroll the preview window down one page

Keybindings can be customized using the `--bind` option in the `FzfTabOptions` field in the color scheme file.

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

4. Image previews

Image previews are available via the `clifmimg` plugin. Run `help image-previews` or consult the Wiki for more information: <https://github.com/leo-arch/clifm/tree/master/misc/tools/imgprev>

11. AUTO-SUGGESTIONS

Gemini is a builtin 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 available (the order can be customized, see below):

- a. ELNs
- b. **clifm** commands and parameters (including the `sel` keyword)
- c. Entries in the command history list (already used commands)
- d. Filenames in the current working directory and paths **(1)**
- e. Entries in the jump database
- f. Aliases names
- g. Bookmarks names
- h. Program names in **PATH**
- i. Shell builtins **(2)**

(1) Fuzzy suggestions are supported. For example: `dwn > Downloads`. Enable this feature via the `--fuzzy-marching` command line switch or setting `FuzzyMatching` to `true` in the configuration file.

(2) The shell name is taken from `/bin/sh`. The following shells 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.

Note: By default, a brief description for internal commands is suggested. You can disable this feature via the `SuggestCmdDesc` option in the configuration file.

To accept the entire suggestion 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 accept the first suggested word only (up to first slash or space), press rather *Alt+Right* or *Alt+f*. Not available for ELNs, aliases and bookmarks names.

Bear in mind that suggestions for ELNs, aliases, bookmarks names, the jump function (invoked by the *j* command), just as filenames and paths (if fuzzy-suggestions are enabled) 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 filename whose ELN is '12', the filename 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 can be customized via the *SuggestionStrategy* option in the configuration file. Each check is assigned a lowercase character:

- a = Aliases names
- c = Possible completions
- e = ELNs
- 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 containing one or more of the above characters. The characters 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 **acefhj** (without quotes). Or, if you prefer to run the history check first: **hacefj**. Finally, you can ignore one or more checks by just omitting the corresponding character (to skip all checks, set the option value to a single dash (-)). So, to ignore the aliases and the ELN checks, set *SuggestionStrategy* to **hcfj**. The default value for this option is **ehfjac**.

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 ELNs, bookmarks names, files in the current directory, and possible completions. Default value: 2;4;36 (dimmed underlined cyan)

sh: Used for entries in the commands history.

sc: Used for aliases and program names in **PATH**.

sr: Used for **clifm** internal commands and parameters.

sp: Greater-than sign (>) used when suggesting ELNs, bookmarks, and aliases names.

You can set *SuggestFiletypeColor* to *true* in the configuration file to use the color of the file type of the current filename (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 it's a broken link) will be used instead of *sf*.

12. SHELL FUNCTIONS

Clifm includes a few shell functions to perform specific actions (cd-on-quit, 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.

13. PLUGINS

Plugins are just scripts or programs (written in any language) intended to add, extend or improve **clifm**'s functionalities. They are linked to actions names defined in a dedicated configuration file (`$XDG_CONFIG_HOME/clifm/profiles/PROFILE/actions.clifm`).

Note: In *stealth mode*, since access to configuration files is not allowed, plugins are disabled.

To list available actions and the plugins they are linked to, run *actions*.

To execute a given plugin, enter the corresponding action name (plus parameters if required).

To get information about a specific plugin, enter the action name followed by `--help`.

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 purpose 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 provided plugins and reading the *actions* command description, just as 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"
```

Plugins can talk to **clifm** via a dedicated pipe created for this purpose and exposed via an environment variable (**CLIFM_BUS**). 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 *file 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.

This is a list of available plugins:

Action name	Description	Dependencies
bn	Create files in batch	–
bcp	Copy files in batch	–
bmi	Import bookmarks	–
clip	Interact with the system clipboard	(1)
unset	Test terminal's colors capability	(2)
cr	Copy files to a remote location	fzf, and scp, ffsend, or croc
da	Disk usage analyzer	du, fzf
dr	Drag and drop files	dragon or dragon-drag-and-drop
fdups	Find/remove file dups	(3)
+	Find files in the current directory	fzf or rofi
_ (underscore)	Quickly change directory	fzf
h	Browse the commands history	fzf
– (yes, just a dash)	Navigate/select/preview files	See section below
*	Select files (includes flat view)	fzf, find
**	Deselect files	fzf
unset	Show git repo status	git (4)
ih	Browse clifm 's manpage	fzf
i	Image thumbnails previewer	sxiv, feh or lsix
++	Jump to a directory in the jump database	fzf or rofi
kd	Decrypt a GnuPG encrypted file	gpg, tar, sed, grep
ke	Encrypt files/dirs using GnuPG	gpg, tar, sed, fzf, awk, xargs
ml	List files by a given MIME type	fzf, file
music	Create a music playlist	mplayer
gg	Pipe files in CWD through a pager	less, column
ptot	Preview PDF files as text	pdftotext
rrm	Recursively remove files	find, fzf
//	Search files by content	fzf, ripgrep
unset	Update plugins	(5)
vid	Preview video files thumbnails	ffmpegthumbnailer
vt	Virtual directory for sets of files	sed
wall	Set image as wallpaper	(6)
unset	Pick/select files via clifm	(7)
Ctrl+y	Copy the line buffer to the clipboard	(8)

(1) xclip or xsel (Xorg), wl-copy/wl-paste (Wayland), clipboard (Haiku), clip (Cygwin), pbcopy/pbget (MacOS), termux-clipboard-get/termux-clipboard-set (Termux), cb (cross-platform: <https://github.com/Slackadays/Clipboard>)

(2) *colors.sh* (by default unset)

(3) find, md5sum, sort, uniq, xargs, sed, stat

(4) The *git_status.sh* 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 (it will be executed immediately before each prompt). Add this line to the main configuration file:

```
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.

(5) *update.sh* (by default unset)

(6) *feh*, *xloadimage*, *hsetroot*, or *nitrogen* (for X); *swww* or *swaybg* (for Wayland)

(7) *file_picker.sh* (by default unset). Usage example: *ls -ld \$(file_picker.sh)*

(8) Dependencies: *cb*, *wl-copy*, *xclip*, *xsel*, *pbcopy*, *termux-clipboard-set*, *clipboard*, or *clip*. Consult the plugin file itself (*xclip.sh*) for more information

Dependencies of the previewer plugin (*fzfnv.sh*)

archives: *atool*, *bsdtar*, or *tar*

images: *kitty terminal*, *imagemagick*, and *ueberzug* or *viu* or *cating* 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*, *links*, *elinks*, or *pandoc*

markdown: *glow*

highlight: *bat*, *highlight*, or *pygmentize*

torrent: *transmission-cli*

json: *python* or *pq*

file info: *exiftool*, *mediainfo*, or *file*

To run the *pager.sh* plugin, for example, you only need to enter the corresponding action name, in this case *gg*. In case of need, all plugins provide a *-h*, *--help* switch for a brief usage description.

Note: The *fzfnv* plugin uses *fzf*(1) to navigate the filesystem and *BFG* (a script located in the plugins directory) to show previews (to display image previews *BFG* requires *ueberzug*(1) or the Kitty protocol via the Kitty terminal). A configuration file (*BFG.cfg*, in the plugins directory itself) is provided to customize the previewer's behavior.

Note 2: An alternative files previewing function (builtin, and thereby faster than *BFG*) is provided by *shotgun*. See the **SHOTGUN** section above for more information.

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

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

14. AUTOCOMMANDS

Heavily inspired by **Vim**, 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 filesystems (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, the *autocommands* function is not restricted to this specific use case: use it for whatever purpose you find useful.

Note: We describe here **permanent** autocommands, which need to be defined in the configuration file. **Temporary** autocommands (set via the command line and valid only for the current directory and the current session) are also available via the *auto* command. See above.

Add a line preceded by the *autocmd* keyword to the main configuration file. The general syntax is:
autocmd TARGET cmd,cmd,cmd

TARGET specifies the object to which subsequent commands will apply. It can match either **directory names (paths)** or **workspaces**.

1. To match directory names use a glob pattern (as specified by **glob(7)**). 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. To match all directories under a specific directory (including this directory itself) use the double asterisk (**). A few examples:

~/Downloads Match exactly the Downloads directory (and *only* this directory) in your home directory
~/Downloads/* Recursively match all subdirectories in ~/Downloads (**excluding** the Downloads directory itself)
~/Downloads/** Recursively match all subdirectories in ~/Downloads (**including** the Downloads directory itself)
~/Downloads/*.d Match all subdirectories in ~/Downloads ending with ".d" (**excluding** the Downloads directory itself)
!~/Downloads Match everything except the ~/Downloads directory

2. You can match workspaces using the ampersand character (@) followed by the *ws* keyword and then the workspace number. For example, to match the third workspace: *@ws3*, or, to match the first workspace, *@ws1*. To match instead all workspaces except the second one: *!@ws2*.

TARGET 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 file list:

Code	Description	Example
cs	Color scheme	cs=zenburn
fc	Files counter	fc=0
ft	Files filter	ft=.*\..pdf\$
fz	Full dir size	fz=1
hf,hh	Hidden files	hf=0
lm	Light mode	lm=1
lv,ll	Long/detail view	lv=0
mf	Max files	mf=100 (1)
mn	Max filename length	mn=20 (1)
od	Only directories	od=1
pg	Pager	pg=0
st	Sort method	st=5
sr	Reverse sort	sr=1

To remove a value, set the option to an empty value. For example, to remove the files filter and the color scheme: *ft=,cs=*.

(1) This option supports the *unset* keyword to remove the corresponding limit. E.g.: *mf=unset,mn=unset*.

Examples

1. Run in light mode and disable the files counter for the *remotes* directory:(1)

```
autocmd /media/remotes/** lm=1,fc=0
```

2. Just a friendly reminder:

```
autocmd ~/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 filenames). Truncate filenames to 20 chars max and run the files pre-viewer:(2)

```
autocmd ~/Downloads mn=20,!~/config/clifm/plugins/fzfnv.sh
```

5. I want the second workspace, no matter what the current directory is, to list files in long view:

```
autocmd @ws2 lv=1
```

6. Mmm, just because I can. Be creative!

```
autocmd /home/user hf=0,cs=nord,lv=1
```

```
autocmd / lv=1,fc=0,cs=solarized,st=5
```

(1) This is the recommended configuration for remote filesystems

(2) As seen here, plugins can be used as well: in this case, we want to run *fzfnv* (to make use of the files preview capability) whenever we enter 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 the *autocommand* function always executes commands as external applications using the system shell.

Bear in mind that *autocmd* directives are evaluated from top to bottom, so that subsequent matching entries will overwrite options set by previous entries.

Autocommand notifications

By default, a gray 'A' is printed to the left of the prompt whenever an autocommand is active for the current directory.

The behavior of this indicator can be customized via the *InformAutocmd* option in the configuration file.

The color code used to colorize this indicator is *ac* (see the **COLORS** section above).

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

To use this feature, you must first set *ReadAutocmdFiles* to *true* in the main configuration file. However, bear in mind that autocommand files will never be read if running on an untrusted environment (i.e. if running with *--secure-cmds*, *--secure-env*, or *--secure-env-full*).

Two files are specifically checked by the autocommands function: *.cfm.in* and *.cfm.out* (they must be non-empty regular files of at most **PATH_MAX** (usually 4096) bytes, and no NUL byte must be contained in them).

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, create both *.cfm.in* and *.cfm.out* files in the home directory with the following content:

For *.cfm.in*:

```
printf "Entering %s ..." "$PWD" && read -n1 && clear
```

For *.cfm.out*:

```
printf "Leaving %s ..." "$OLDPWD" && read -n1
```

15. FILE TAGS

Etiqueta is **clifm**'s builtin files tagging system

1. How does Etiqueta work?

File tags are created via symlinks using a 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 [add, del, list, list-full, new, merge, rename, untag] [FILE]... [:]TAG
```

NOTE: The *:TAG* notation is used for commands taking both file and tag names: `'tag add 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 del png'`, to delete the tag named *png*.

Both short and long command format can be used:

Short format	Long format	Description
ta	tag add	Tag files
td	tag del	Delete tag(s)
tl	tag list	List tags or tagged files
tm	tag rename	Rename tags
tn	tag new	Create new tag(s)
tu	tag untag	Untag file(s)
ty	tag merge	Merge two tags

3. Usage examples

Short format	Long format	Description
tl	tag list	List available tags
–	tag list–full	List available tags and all tagged files
tl work	tag list work	List all files tagged as <i>work</i>
tl file.txt	tag list file.txt	List tags applied to the file <i>file.txt</i>
tn dogs cats	tag new dogs cats	Create two empty tags: <i>dogs</i> and <i>cats</i>
ta *.png :images :png	tag add *.png :images :png	Tag PNG files as both <i>images</i> and <i>png</i> (1) (2)
ta sel :special	tag add sel :special	Tag all selected files as <i>special</i>
tr documents docs	tag rename documents docs	Rename the tag <i>documents</i> as <i>docs</i>
ty png images	tag merge png images	Merge the tag <i>png</i> into <i>images</i> (3)
td images	tag del images	Remove the tag <i>images</i> (4)
tu :work file1 dir2	tag untag :work file1 dir2	Untag a few files from <i>work</i> (5)

(1) Tags are created if they do not exist

(2) Since *add* is the default action, it can be omitted: *tag *.png :images :png*.

(3) All files tagged as *png* will be now tagged as *images*, and the *png* tag will be removed.

(4) Untag all files tagged as *images* and remove the tag itself

(5) Tab completion is available to complete tagged files. If using the *fzf* mode, multiple files can be selected using the 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:

Command	Description
p t:docs	Print properties of files tagged as <i>docs</i>
r t:images	Remove all files tagged as <i>images</i>
stat t:docs t:work	Run <i>stat(1)</i> over all files tagged as <i>docs</i> and all files tagged as <i>work</i>

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 mark the two files you need using **TAB**, press **Enter** or **Right**, and the full path to 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**'.

16. VIRTUAL DIRECTORIES

CLiFM is able to read and list files from the standard input stream (STDIN). Each file 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 is stored as a symbolic link pointing to the original file in a temporary directory (called here virtual directory) with read-only (0500) permissions. This directory, and all its contents, will be deleted at program exit. Use the `--virtual-dir` command line flag to specify a custom directory (it if does not exist, it will be created) instead of the default one, created in the system temporary directory (usually `/tmp/clifm/USER/vdir.XXXXXX`, where XXXXXX is a random six digits string).

The user can operate on these files as if they were any other regular file, since **all operations performed on these symbolic links** (provided the current working directory is the virtual directory where all these files are stored) **are performed on the target files and NOT on the symbolic links themselves**.

Once in the virtual directory, files are listed by default using only the base name of the target file. For example, if the target file is `/home/user/Downloads/myfile.tar.gz`, this file will be listed as `myfile.tar.gz`. If this file already exists in the virtual directory (because there is another target file with the same base name, say, `/home/user/Documents/tars/myfile.tar.gz`), a random six digits suffix will be appended to the file (for instance, `myfile.tar.gz.12Rgj6`).

Since this listing mode does not allow the user to get a clear idea of the actual location of each listed file, a keybinding (by default `Alt+w`) is available to toggle short (base names only) and long filenames: in this latter case, filenames are listed using the full path to the target file, replacing slashes by colons (:). For example, if the target file is `/home/user/Downloads/myfile.tar.gz`, it will be listed in the virtual directory as `home:user:Downloads:myfile.tar.gz`.

If you prefer the long names approach, you can use the `--virtual-dir-full-paths` command line flag.

Note: Bear in mind that the restore last path function is disabled when listing in this way.

CLiFM provides two ways of using virtual directories:

1. Reading files from the standard input
2. Listing sets of files via the `virtualize.sh` plugin (which is in fact a special use case of point 1)

1. Standard input

Examples:

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

This command will pass all files in the directory `/var` to CLiFM

If you need to perform more specific queries, you can use `find(1)` 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
```

2. The virtualization plugin

The *virtualize.sh* plugin, bound by default to the *vt* action name, is intended to provide an easy way of listing sets or collections of files, such as selected, tagged, or bookmarked files. For example, to send all selected files to a virtual directory, you can issue this command:

```
vt sel
```

and, if you want rather files tagged as PDF:

```
vt t:PDF
```

Of course, individual files can also be used:

```
vt file1 file2 file3
```

Once executed, the *vt* plugin will launch a new instance of CliFM (on a new terminal emulator window) where you can operate on the specified files as if they were just normal files. Once done, quit this new instance (via the *q* command) to return to the primary instance of CliFM.

Note: By default, the terminal emulator used is **xterm**(1), but it can be changed by editing the plugin script (*virtualize.sh*).

If navigating the filesystem, you can quickly go back to the virtual directory using the *-d* option: *vt -d*. The navigation keys (see the **KEYBOARD SHORTCUTS** section above) and the **CLIFM_VIRTUAL_DIR** environment variable are also available (*Shift+Left/Shift+Right* or *cd \$CLIFM_VIRTUAL_DIR*).

Tip: Write an alias to make this even easier:

```
alias vtd='cd $CLIFM_VIRTUAL_DIR'
```

17. 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 this function (this does not apply, however, to the long view: since we need to display files information, **fstatat**(3) is required).

When running in light mode, however, a few features are lost:

1. 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 filesystem not

returning the *d_type* value, for example, loop devices), **clifm** will fall back to **stat(3)** to get basic files classification.

2. Color per file extension is disabled for performance reasons.

3. The marker for selected files (*) is lost as well: to keep track of selected files and thus recognize them in the current list of files, we make use of files device and inode number, which is provided by **fstatat(3)**.

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 builtin 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)**.

18. KANGAROO FREQUENCY 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) Permanent entry ('+'): this directory will not be removed from the database, no matter its rank
- b) Number of visits
- c) Date of first visit (seconds since the Unix epoch)
- d) Date of the last visit (seconds since Unix epoch)
- e) The full path to each visited directory

With this information it is possible to build 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 **freccency** 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 * VISIT-CREDIT) / 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*).

NOTE: VISIT-CREDIT is a hardcoded value: 200.

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 users 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, directories active in a **workspace** 300 credits, and directories marked as **permanent** 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, pinned, and marked as permanent, it gets 1800 extra credits. In this way the total rank of this directory in the matching process is 2900. In doing so, 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) If *PurgeJumpDB* is set to *true* (see the main configuration file), each entry in the database is checked at startup to remove non-existent directories. This option is set to *false* by default to avoid removing paths pointing to unmounted filesystems (like removable devices or remote locations) which you still might want to keep. Non-existent directories, however, will be removed soon or later anyway due to their low rank value (see below).

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. If set to 0, by contrast, directories will never be removed from the database.

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* (and provided this latter is not 0), 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, just as pinned, bookmarked directories, and directories currently used in some workspace, will not be removed from the database, no matter what their rank is. In other words, if you want to indefinitely keep a given directory in the jump database, bookmark it, or mark it as permanent (edit the database, via *je* or *j --edit*, and prepend a plus sign (+) to the corresponding entry).

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**.

19. 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**

CLICOLOR_FORCE

Force the use of colors, even if the terminal does not report color support

CLIFM_FORCE_COLOR

Same as **CLICOLOR_FORCE**, but accepts a specific color value (8, 16, 256, truecolor or 24bit).

COLORTERM

If set to either *truecolor* or *24bit*, **clifm** assumes the terminal emulator to be capable of displaying true colors (and thereby also 256 colors), despite what the **terminfo**(5) database informs.

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 elements of **clifm**'s interface

CLIFM_DATE_SHADES

A comma separated list of colors used to print timestamps based on age

CLIFM_SIZE_SHADES

Same as **CLIFM_DATE_SHADES**, but for file sizes

CLIFM_PREVIEW_MAX_SIZE

If running with *--preview*, or *PreviewMaxSize* is not set in the configuration file, no preview is generated for files larger than this value. The value must be specified in KiB: for example, 2048 is read as 2 MiB.

CLIFM_TEMPLATES_DIR

A custom file templates directory

CLIFM_HISTFILE

A custom commands history file

CLIFM_FILTER

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

CLIFM_SUDO_CMD

Name of the authenticator program. Used by the *X* command (to launch a new instance of ****clifm**** as root), the *Alt+v* keybinding (to prepend the authenticator program name to the current command line), and for some operations on archives (ISO files). Defaults to *sudo* (or *doas* if compiled on OpenBSD). Example: *CLIFM_SUDO_CMD=doas clifm*.

SHELL

An absolute path to the shell to be used by **clifm** to run external commands. Only values found in */etc/shells* are allowed.

CLIFM_SHELL

Same as **SHELL**, but specific to **clifm** (takes precedence over **SHELL**).

TMPDIR

Path to a directory where temporary files will be created

CLIFM_TMPDIR

Same as **TMPDIR**, but specific to **clifm** (takes precedence over **TMPDIR**)

TERM

Terminal type for which output is to be prepared

FZF_DEFAULT_OPTS

A quoted list of options to be passed to *fzf* (if used for tab completion)

TIME_STYLE

If set from neither *--time-style* nor *TimeStyle* (in the configuration file), use this time style for the long view mode

CLIFM_TIME_STYLE

Same as **TIME_STYLE**, but specific to **clifm** (takes precedence over **TIME_STYLE**)

PTIME_STYLE

If set from neither `--ptime-style` nor `PTimeStyle` (in the configuration file), use this time style for the `p/pp` command and the `--stat/--stat-full` command line switches

CLIFM_COLUMNS

The number of terminal columns

CLIFM_LINES

The number of terminal lines

CLIFM_MIMETYPES_FILE

Set a custom mime.types file (instead of the default, `~/mime.types`). Consult the **FILE OPENER** section form more information.

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

CLIFM

Path to 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 to the configuration directory: `cd $CLIFM` or just `$CLIFM`

CLIFMRC

Path to the main configuration file (by default `~/config/clifm/profiles/default/clifmrc`)

CLIFM_PID

PID number of **clifm**'s running instance

CLIFM_VERSION

Version number of **clifm**'s running instance

CLIFM_VIRTUAL_DIR

Path to the currently used virtual directory only if (and while) the virtual directory function is executed. See the **VIRTUAL DIRECTORIES** section above.

SHLVL

Incremented by one each time a new shell is started.

CLIFMLVL

Same as **SHLVL**, but specific to **clifm**.

If *Notifications* is set to *false* for the current prompt, 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_ERROR_MSGS

Current amount of error messages

CLIFM_STAT_WARNING_MSGS

Current amount of warning messages

CLIFM_STAT_NOTICE_MSGS

Current amount of notice 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

When running a plugin, the following environment variables are set:

CLIFM_BUS

The path to a pipe by means of which plugins can talk to **clifm**. See the **PLUGINS** section for more information.

CLIFM_COLOR_SCHEME

Set to the name of the current color scheme

CLIFM_COLORLESS

Set to 1 if running without colors

CLIFM_CUR_WS

Set to the current workspace number

CLIFM_DIRS_FIRST

Set to 1 if list-dirs-first is enabled. Otherwise it is set to 0

CLIFM_FILES_COUNTER

Set to 1 if files-counter is enabled. Otherwise it is set to 0

CLIFM_FILES_FILTER

Set to the files filter string. Unset if no files filter is set

CLIFM_FILTER_REVERSE

Set to 1 if filter-reverse is set or to 0 otherwise (unset if no files filter is set)

CLIFM_FOLLOW_LINKS

Set to 1 if follow-symlinks is enabled. Otherwise it is set to 0

CLIFM_LIGHT_MODE

Set to 1 if light-mode is enabled. Otherwise it is set to 0

CLIFM_LINE

When running a plugin via a keybinding, this variable holds the content of the current line buffer. For a usage example see the *xclip.sh* plugin.

CLIFM_LONG_VIEW

Set to 1 if long-view is enabled. Otherwise it is set to 0

CLIFM_MAX_FILES

Set to MAX_FILES if max-files is set. Unset otherwise.

CLIFM_ONLY_DIRS

Set to 1 if only-dirs is enabled. Otherwise it is set to 0

CLIFM_PLUGINS_HELPER

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

CLIFM_PROFILE

Set to the name of the current profile.

CLIFM_SEL_FILES

Set to the number of currently selected files (unset if there are no selected files)

CLIFM_SELFILE

Set to the path to the current selection file

CLIFM_SHOW_HIDDEN

Set to 1-3 if show-hidden is enabled (true, first, last, in this order). Otherwise it is set to 0.

CLIFM_SORT_REVERSE

Set to 1 if sort-reverse is enabled. Otherwise it is set to 0

CLIFM_SORT_STYLE

Set to the current sorting method

CLIFM_TRASH_FILES

Set to the number of currently trashed files (unset if there are no trashed files)

CLIFM_TRUNCATE_NAMES

Set to 1 if truncate-names is enabled. Otherwise it is set to 0

20. 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 at protecting 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, aliases, plugins, (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" (space, new line char, and horizontal TAB)

As a plus, **1)** core dumps are disabled, **2)** the umask value is set to *0077* at startup and the creation mode (when using the *new* command) is forced to *0700* for directories and *0600* for files, **3)** non-standard file descriptors (>2) are closed, **4)** SUID/SGID privileges, if any, are dropped, and **5)** autocommand files aren't read at all (even if *ReadAutocmdFiles* is set to true).

b) Full: this mode is just like the regular mode, except that *nothing* is imported from the environment at all and only **PATH**, **HOME**, **USER**, **SHELL**, 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*), aliases, and plugins, 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 four different, though intrinsically related tasks aimed to mitigate command injection and/or unexpected behavior:

a) Plugins are disabled.

b) 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 **d** below) will be executed. This is done in order to prevent the execution of custom binaries/scripts, for example: */tmp/exec_file*.

c) 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).

d) A secure environment is set (`--secure-env` is implied; to run on a fully sanitized environment run as follows: `--secure-cmds --secure-env-full`).

21. 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 ELNs 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. It can run any external command as well, provided external commands are allowed (see the `-x` option, the `ext` command, or the configuration file).

External commands are executed using an actual shell (say, `/bin/bash`), which is specified by **clifm** as follows:

1. If the **CLIFM_SHELL** environment variable is set, this value is used.
2. If the **SHELL** environment variable is set, this value is used.
3. If none of the above, the value will be taken from the `passwd` database (via `getpwuid(3)`).

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.

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 filename `"ñandú"` (the Spanish word for 'rhea'), it will be correctly displayed by the Linux console, Lxterminal, and Urxvt, but not thus by more basic terminal emulators like Aterm.

Spaces and filenames:

When dealing with filenames containing spaces, you can use both single and double quotes (e.g.: `"this file"` or `'this file'`) plus the backslash character (e.g.: `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. E.g.: `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 using the appropriate command line options.

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).

22. FILES

CONFIGURATION FILE

The main configuration file is looked up in these places (and in this order):

1. `-c, --config-file` switch
2. `$CLIFMRC` variable
3. `$XDG_CONFIG_HOME/clifm/profiles/PROFILE/clifmrc` directory

If `$XDG_CONFIG_HOME` is not set, `$HOME/.config` is used instead. If running with `secure-environment` (via either `--secure-cmds`, `--secure-env`, or `--secure-env-full`) no environment

variable is read, so that the home directory is taken instead from the password database (via **getpwuid(3)**).

PROFILE is by default *default* (unless set via *-P,--profile*).

You can access the configuration file either via the *config* command or pressing F10.

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.clifm*. In this file you can add those commands you want to be executed at startup. You can also permanently set here some custom variables, e.g.: *'dir="/path/to/dir"'*. This variable may be used as a shortcut to that directory, for instance: *cd \$dir*. Custom variables could also be temporarily defined via the command prompt: E.g.: *user@hostname ~ \$ var="This is a test"*. Temporary variables will be removed at program exit. Internal variables are disabled by default; enable them via the *--int-vars* command line switch.

PROMPTS FILE

This file contains prompts definitions and is located in *DATADIR/clifm/prompts.clifm*. It will be copied automatically to *\$XDG_CONFIG_HOME/clifm/prompts.clifm* if it doesn't exist. The *Prompt* line in the color scheme file should point to one of the prompt names defined in this file. See the **PROMPT** section for more information.

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 (or just themes) as files with a *.clifm* extension. You can copy these themes to the local colors directory (*\$XDG_CONFIG_HOME/clifm/colors*) and edit them to your liking (or create new themes from the ground up). Themes in the local colors directory take precedence over those in the system directory. You can create as many themes as you want by dropping them into the local colors directory. The default color scheme file (*default.clifm*) can be used as a guide.

ACTIONS FILE

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

PREVIEW FILE

The preview file is *\$XDG_CONFIG_HOME/clifm/profiles/PROFILE/preview.clifm* and is shotgun's configuration file. It makes use of the same syntax used by the mimelist file. It will be copied from *DATADIR/clifm* (usually */usr/share/clifm*).

BOOKMARKS FILE

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

HISTORY FILE

The history file is `~/config/clifm/profiles/PROFILE/history.clifm`. 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/cmdlogs.clifm`. Command logs keep track of commands entered in the command line. These logs have this form: "[date] current_working_directory:command".

MESSAGES LOG FILE

The messages log file is `$XDG_CONFIG_HOME/clifm/profiles/PROFILE/msglogs.clifm`. Message logs are a record of errors and warnings and have the following form: "[date] message".

KANGAROO DATABASE

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

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

23. EXAMPLES

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

NOTE 2: Suggestions for possible completions are printed next to the text typed so far. To accept the given suggestion press *Right* (or *Alt+f* to accept only the first/next suggested word). Otherwise, the suggestion is just ignored

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

1. NAVIGATION

Command	Description
/etc	Change directory to <i>/etc</i> (1)
5	Change to the directory whose ELN is 5 (2)
j <TAB> (also dh <TAB>)	Navigate through visited directories
j xproj	Jump to <code>~/media/data/docs/work/mike/xproject</code> (3)
b (Shift+Left, Alt+j)	Go back in the directory history list
f (Shift+Right, Alt+k)	Go forth in the directory history list
.. (Shift+Up, Alt+u)	Change to the parent directory
...	Change to the parent directory of the current parent directory (4)
bd w	Change to the parent directory matching "w" (5)
ws2 (Alt+2)	Switch to the second workspace (6)
/*.pdf<TAB>	List PDF files (current dir)
=x<TAB>	List executable files (current dir) (7)
@gzip<TAB>	List files (current dir) whose MIME type includes "gzip"
pin mydir	Pin the directory named <i>mydir</i>
,	Change to pinned directory
view (Alt+-)	Preview files (current dir) (8)
pg (Alt+0)	Run MAS, the files pager, on the current directory

- (1) `cd /etc` also works
- (2) Press TAB to make sure 5 is the file you want, or just pay attention to the suggestion. Press Right to accept the given suggestion
- (3) This depends on the database ranking. For more accuracy: `j mike xproj`. Tab completion is available: `j xproj<TAB>`
- (4) This is the *fastback* function: each subsequent dot after the two first dots is understood as an extra `"/.."`
- (5) Type `bd <TAB>` to list all parent directories
- (6) `Alt+[1-4]` is available for workspaces 1-4
- (7) Type `=<TAB>` to get the list of available file type characters. Consult the **FILE FILTERS** section above for more information
- (8) This feature depends on `fzf(1)`

2. FILE OPERATIONS

Command	Description
<code>myfile.txt</code>	Open <i>myfile.txt</i> (with the default associated application)
<code>myfile.txt vi</code>	Open <i>myfile.txt</i> using vi (1)
<code>24&</code>	Open the file whose ELN is 24 in the background
<code>n myfile mydir/</code>	Create a new file named <i>myfile</i> and a new directory named <i>mydir</i> (2)(3)
<code>p4</code>	Print the properties of the file whose ELN is 4
<code>pc myfile.txt</code>	Edit the permission set of the file <i>myfile.txt</i> (use <i>oc</i> to edit ownership)
<code>s *.c</code>	Select all c files in the current directory
<code>s /media/*<TAB></code>	Interactively select files in the directory <i>/media</i> (4)
<code>s 1-4 8 19-26</code>	Select multiple files in the current directory by ELN
<code>sb (sel<TAB> or s:<TAB>)</code>	List selected files (5)
<code>ds (ds <TAB>)</code>	Selectively deselect files using a menu
<code>bm add mydir/ mybm</code>	Bookmark the directory <i>mydir/</i> as "mybm"
<code>bm mybm (b:mybm)</code>	Access the bookmark named <i>mybm</i> (6)
<code>bm del mybm</code>	Remove the bookmark named <i>mybm</i>
<code>bm (Alt+b or b:<TAB>)</code>	Open the bookmark manager
<code>t 1-3 *.old</code>	Trash a few files
<code>u (u <TAB>)</code>	Selectively undelete/restore trashed files using a menu
<code>t del (t del <TAB>)</code>	Selectively remove files from the trash can using a menu
<code>t empty</code>	Empty the trash can
<code>ta *.pdf :mypdfs</code>	Tag all PDF files in the current directory as <i>mypdfs</i>
<code>p t:mypdfs</code>	Print the file properties of all files tagged as <i>mypdfs</i>
<code>/* .pdf</code>	Search for all PDF files in the current directory
<code>c sel</code>	Copy selected files to the current directory
<code>c *.txt 2</code>	Copy all txt file to the directory whose ELN is 2
<code>r sel</code>	Remove all selected files (7)
<code>m4</code>	Rename the file whose ELN is 4 (8)

- (1) Use the *ow* command to select the opening application from a menu: `ow myfile.txt` or `ow myfile.txt <TAB>`
- (2) Note the ending slash in the directory name
- (3) Since **clifm** is integrated to the system shell, you can also use any of the shell commands you usually use to create new files. E.g.: `touch myfile` or `nano myfile`
- (4) Only for non-standard tab completion: `fzf`, `fnf`, `smenu`
- (5) You can also TAB expand the *sel* keyword: `p sel<TAB>` to list selected files (and optionally mark

multiple selected files to operate on)

- (6) Type *bm* <TAB> to get the list of available bookmark names
- (7) To remove files in bulk use the *rr* command
- (8) To rename files in bulk use the *br* command

3. MISC

Command	Description
hh (Alt+.)	Toggle hidden files
ll (Alt+l)	Toggle detail/long view mode
rf (Enter –on empty line– or Ctrl+l)	Clear/refresh the screen
Alt+,	Toggle list-directories-only
Alt+Tab, Ctrl+Alt+i	Toggle disk usage analyzer mode
!<TAB>	Navigate through the command history
config (F10)	View/edit the main configuration file
pf set test	Change to profile <i>test</i>
actions	List available actions/plugins
icons on	Want icons?
cs (cs <TAB>)	List available color schemes
prompt (prompt <TAB>)	List available prompts
q	I'm tired, quit

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

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.

BUG AND FEATURE REQUESTS

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

AUTHOR

L. M. Abramovich <leo.clifm@outlook.com>

For additional contributors, use *git shortlog -s* on the `clifm.git` repository.