ansicolortags Documentation

Release public

Lilian Besson

09/08/2017, 15h:27m:32s

Contents

1 Installation **1**
- 1.1 Dependencies . . . 1
- 1.2 How to install it ? . . . 1
  - 1.2.1 1. Directly, with pip . . . 2
  - 1.2.2 2. Download, extract, and use `setup.py` . . . 2
  - 1.2.3 3. *Without installing it* . . . 2

2 Examples **2**
- 2.1 The function `ansicolortags.printc()` . . . 2

3 The script `ansicolortags.py` **3**
- 3.1 `python -m ansicolortags --help` . . . 3
- 3.2 `python -m ansicolortags --test` . . . 4
- 3.3 `--ANSI` or `--noANSI` option . . . 4
- 3.4 `python -m ansicolortags --generate` . . . 5

4 Complete documentation **6**
- 4.1 Contact the author if needed? . . . 6

5 Index and tables **7**
- 5.1 Table of contents . . . 7
  - 5.1.1 Installation . . . 7
    - 5.1.1.1 Dependencies . . . 7
    - 5.1.1.2 How to install it ? . . . 7
      - 5.1.1.2.1 1. Directly, with pip . . . 7
      - 5.1.1.2.2 2. Download, extract, and use `setup.py` . . . 7
      - 5.1.1.2.3 3. *Without installing it* . . . 8
  - 5.1.2 ansicolortags module . . . 8
    - 5.1.2.1 List of functions . . . 8
      - 5.1.2.1.1 To print a string . . . 8
      - 5.1.2.1.2 To clean the terminal or the line . . . 9
      - 5.1.2.1.3 Others functions . . . 9
    - 5.1.2.2 Example of use (module) . . . 9
    - 5.1.2.3 Example of use (script) . . . 10
    - 5.1.2.4 Auto detection . . . 10
    - 5.1.2.5 Elsewhere online . . . 10
    - 5.1.2.6 Copyrigth . . . 10

5.1.2.7 Complete documentation 10
5.1.3 TODOs for ansicolortags package ? 14
5.2 The MIT License (MIT) 14
5.3 Cloud of words 15

Python Module Index **16**

Welcome to the documentation for `ansicolortags`, a Python 2 or 3 module to use ANSI colors in a terminal application from Python.

The `ansicolortags` module provides an efficient and useful function (`printc()`) to print colored text in a terminal application with Python 2 and 3, with a *HTML-tag* like style:

```
>>> from ansicolortags import printc  # Import the function
>>> printc("I like my sky to be <blue>blue<reset>, not <black>dark<reset> !")
I like my sky to be blue, not dark !
```

- This project is open-source here on BitBucket.
- This project is also available from Pypi, so a quick overview and the last release can be downloaded from Pypi: https://pypi.python.org/pypi/ansicolortags !

Installation

TL;DR: install this module and script in one command with pip:

```
pip install ansicolortags
```

Dependencies

The project is entirely written in pure Python, supporting both version 2 (2.7+) and version 3 (3.1+). It has **no dependencies** except a Python interpreter! For more details about the Python language, see its official website.

How to install it ?

There is three different ways:

1. Directly, with pip

The preferred way is to install `ansicolortags` directly with `pip` (what is pip?):

```
$ pip install ansicolortags
```

On GNU/Linux, it might be necessary to give it *sudo rights*:

```
$ sudo pip install ansicolortags
```

If you have both Python 2 and 3, and if you want to use the module *from both*, be sure to install it *with both* `pip2` and `pip3` :

```
$ sudo pip2 install ansicolortags  # For Python 2 (v2.7+)
$ sudo pip3 install ansicolortags  # For Pythoh 3 (v3.1+)
```

2. Download, extract, and use `setup.py`

Or you can also follow these 4 steps:

1. download the file ansicolortags.tar.gz from Bitbucket, or from PyPi (if needed, it is signed numerically with my PGP key);
2. extract it (with `tar xzfv`, or a graphical solution, like File Roller);
3. then go in the subdirectory (`cd ansicolortags-0.4/`);
4. and finally install it with Python distutils `setup.py` tool:

   ```
   $ python setup.py install
   ```

   Or maybe with sudo rights, if the first try did not work :

   ```
   $ sudo python setup.py install
   ```

   For more information, run `python setup.py help` or `python setup.py install help`.

3. *Without installing it*

Note that installation *is not mandatory* : a third solution is to simply include JUST the file `ansicolortags.py`, and embed it in your own projects. The project can be used without installing *anything elsewhere*.

Examples

The function `ansicolortags.printc()`

The main function of this module is `printc` (*`ansicolortags.printc()`* (page 12)), for example use it like `printc("my string with color tags")`. This function works *exactly* like `print("my string with color tags")`.

For instance, a quick description of super hero's costumes can be done like this:

```
>>> printc("<reset><white>Batman's costume is <black>black<white>, Aquaman's costume␣
↪is <blue>blue<white> and <green>green<white>, and Superman's costume is <red>red
↪<white> and <blue>blue<white> ...<reset>")
Batman's costume is black, Aquaman's costume is blue and green, and Superman's␣
↪costume is red and blue ...
```

(Sorry, but it is hard to embed colors in the output of a Python command in a Sphinx generated web-pages.)

Another example, it will print the text *"France flag is blue, white and red !"* with appropriate colors:

```
>>> from ansicolortags import printc  # Import the function
>>> printc("France flag is <blue>blue<reset>, <white>white<reset>, and <red>red<reset>
↪ !")
France flag is blue, white and red !
```

The function `printc()` should be imported from *ansicolortags* (page 8) (*ansicolortags.printc()* (page 12)).

Note that other useful functions are defined: *ansicolortags.writec()* (page 13) to write to a file, *ansicolortags.xtitle()* (page 14) to change the title of the terminal, *ansicolortags.sprint()* (page 12) to convert all the tags (e.g., `<red>`) in a string to their ANSI Code value (e.g., `\033[01;31m`), etc.

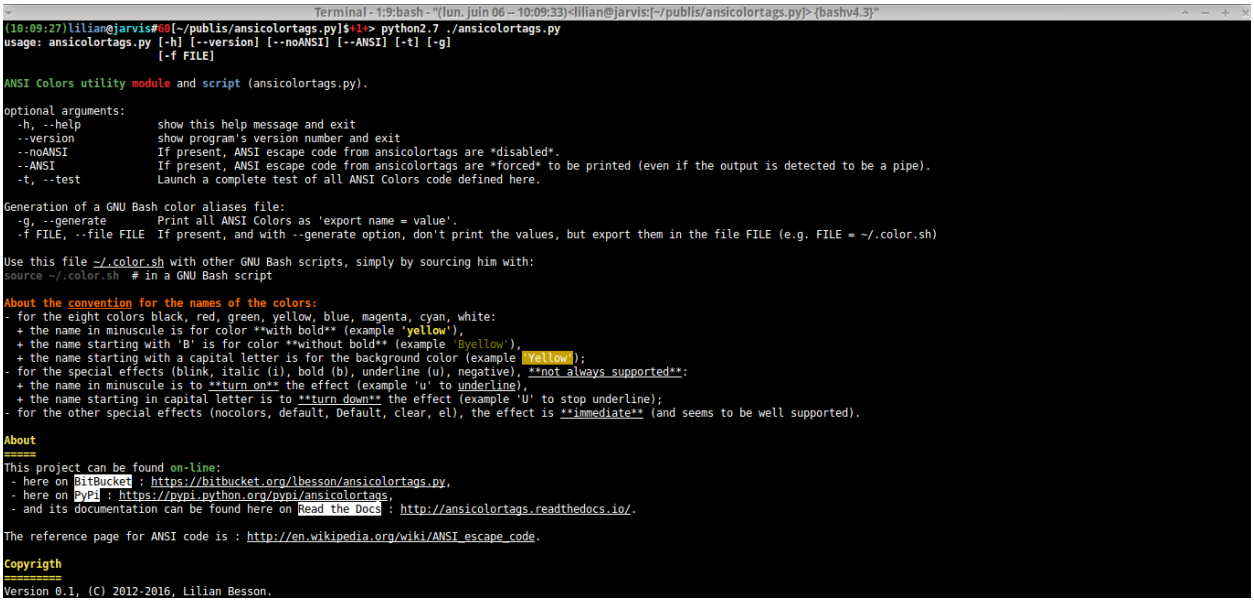
The script `ansicolortags.py`

But the project also installs a script, `ansicolortags.py`, which can be executed directly, or called with `python -m ansicolortags` after installation.

The script itself does not have a useful purpose, except for testing and demonstrating the capabilities of this project. If needed: `ansicolortags.py`.

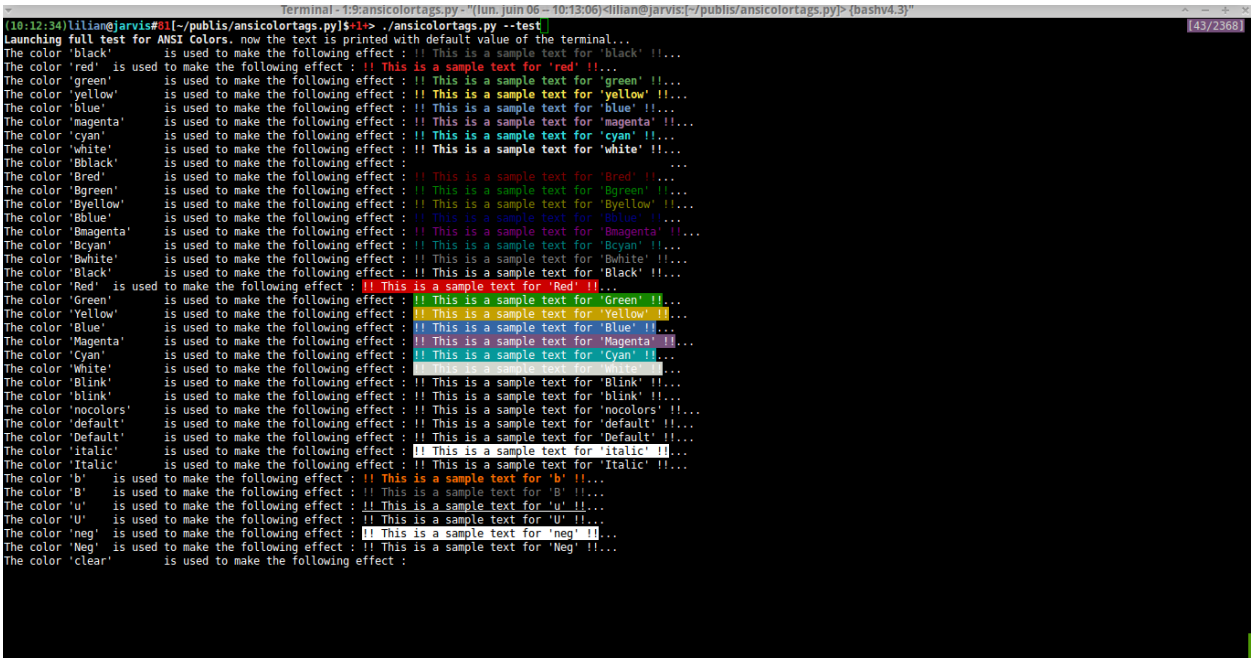
`python -m ansicolortags --help`

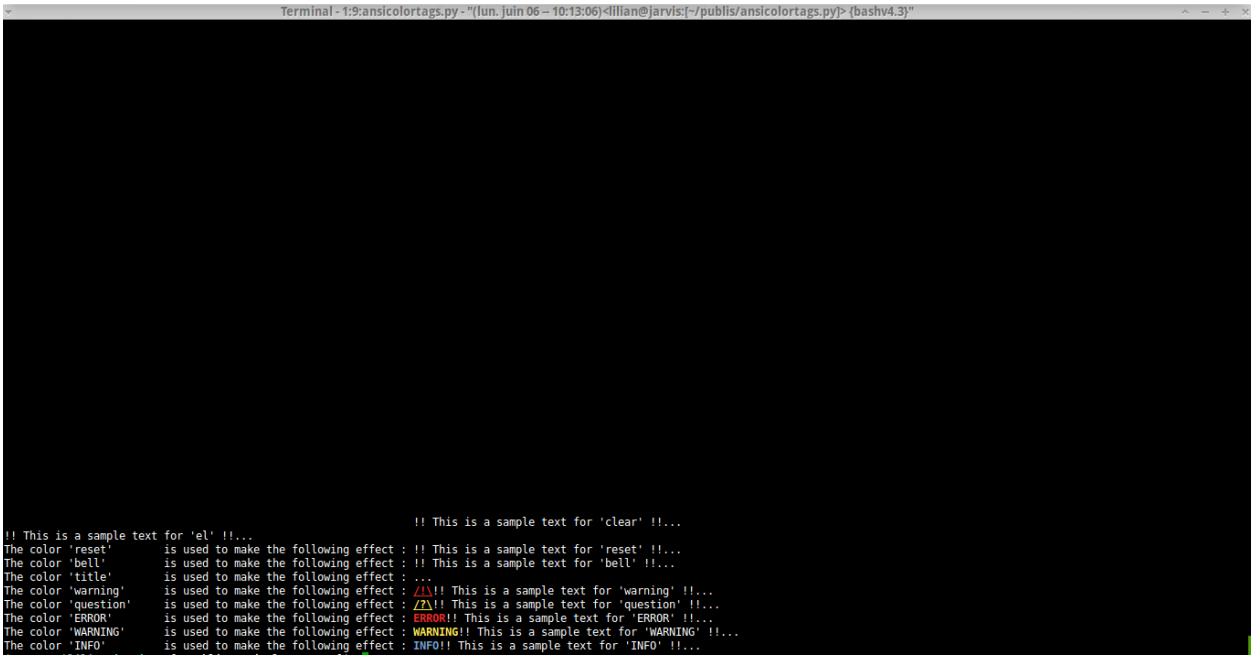
This command shows the help of the script, colored with the tools defined in the script:



`python -m ansicolortags --test`

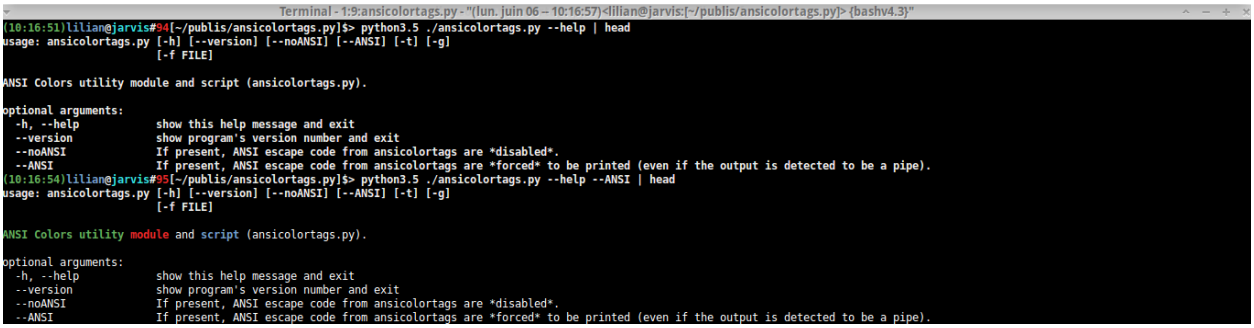
This command shows a complete test of all tags defined in the module:

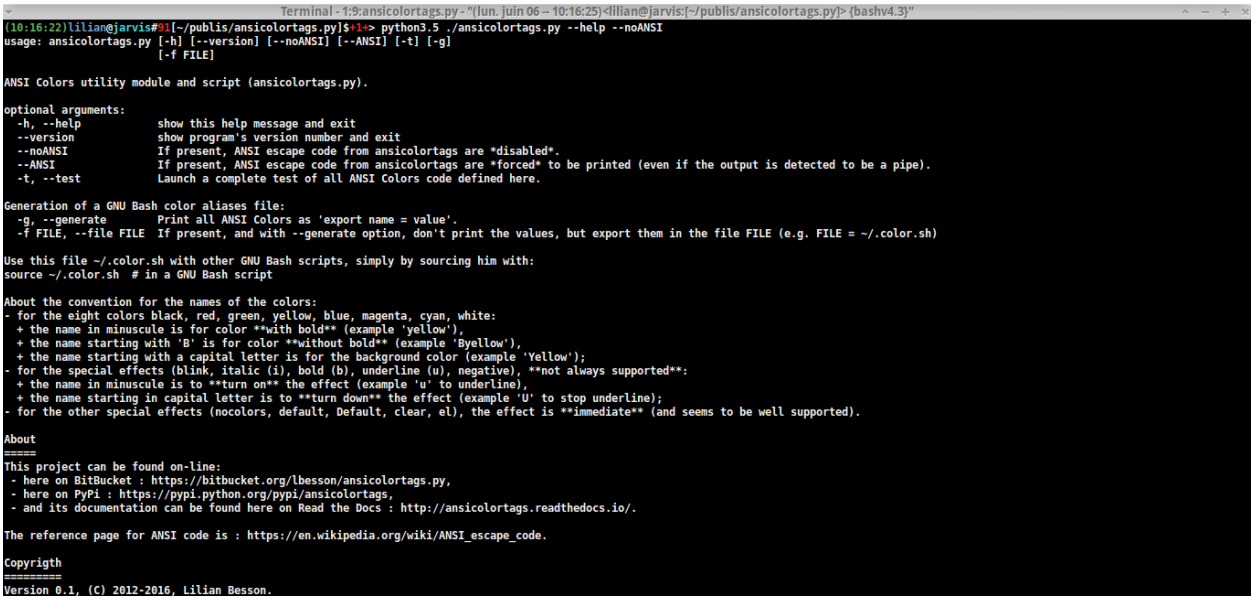




`--ANSI` or `--noANSI` option

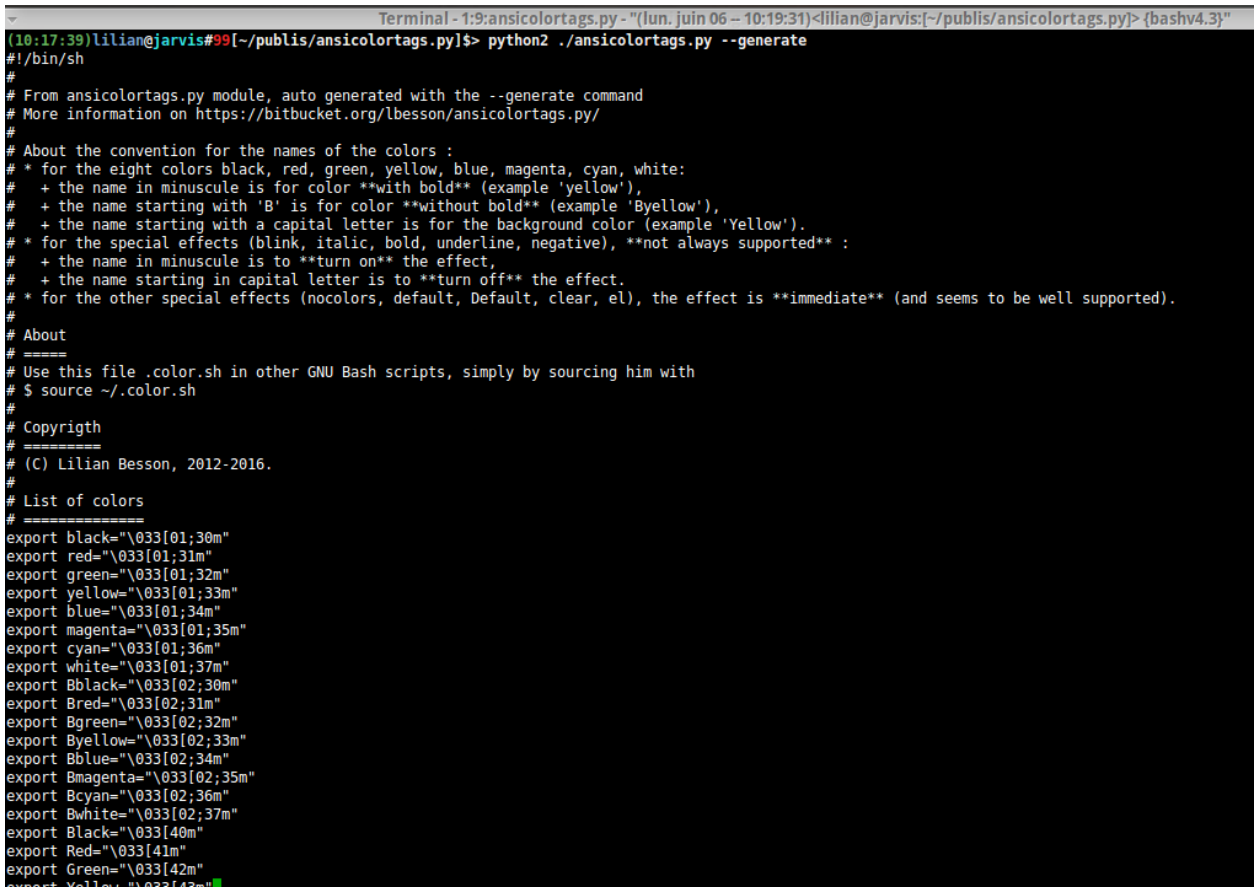
You can force to use ANSI codes (even if they appear to not be supported be the output, e.g. a pipe) with the `--ANSI` flag option, or force to disable them with the `--noANSI` flag option:





`python -m ansicolortags --generate`

This command can be used to generate a .color.sh file, to be used in any GNU Bash script:



Complete documentation

And, a detailed description of every functions and every constants of the *ansicolortags* (page 8) module is available on the documentation of the module ansicolortags (automatically generated from the docstrings in the file).

Contact the author if needed?

Hi, I am Lilian Besson, a French student at ÉNS de Cachan, in Mathematics and computer science (CS).

If needed, feel free to contact me :

1. either with this web page;
2. or via my bitbucket account lbesson;
3. or via email here (remove the [] and change DOT to . and AT to @.

You can use this form to inform me of a bug on `ansicolortags.py`: https://bitbucket.org/lbesson/ansicolortags.py/issues?status=new&status=open !

Index and tables

- genindex
- modindex
- search

Table of contents

Installation

TL;DR: install this module and script in one command with pip:

```
pip install ansicolortags
```

Dependencies

The project is entirely written in pure Python, supporting both version 2 (2.7+) and version 3 (3.1+). It has **no dependencies** except a Python interpreter! For more details about the Python language, see its official website.

How to install ?

There is three different ways:

1. Directly, with pip

The preferred way is to install `ansicolortags` directly with `pip` (what is pip?):

```
$ pip install ansicolortags
```

On GNU/Linux, it might be necessary to give it *sudo rights*:

```
$ sudo pip install ansicolortags
```

If you have both Python 2 and 3, and if you want to use the module *from both*, be sure to install it *with both* `pip2` and `pip3`:

```
$ sudo pip2 install ansicolortags  # For Python 2 (v2.7+)
$ sudo pip3 install ansicolortags  # For Pythoh 3 (v3.1+)
```

2. Download, extract, and use `setup.py`

Or you can also follow these 4 steps:

1. download the file ansicolortags.tar.gz from Bitbucket, or from PyPi (if needed, it is signed numerically with my PGP key);
2. extract it (with `tar xzfv`, or a graphical solution, like File Roller);
3. then go in the subdirectory (`cd ansicolortags-0.4/`);

4. and finally install it with Python distutils `setup.py` tool:

```
$ python setup.py install
```

Or maybe with sudo rights, if the first try did not work :

```
$ sudo python setup.py install
```

For more information, run `python setup.py help` or `python setup.py install help`.

3. *Without installing it*

Note that installation *is not mandatory* : a third solution is to simply include JUST the file `ansicolortags.py`, and embed it in your own projects. The project can be used without installing *anything elsewhere*.

ansicolortags module

An efficient and simple ANSI colors module (and also a powerful script), with functions to print text using colors. https://bitbucket.org/lbesson/ansicolortags.py

The names of the colors follow these conventions:

- for the eight ANSI colors (black, red, green, yellow, blue, magenta, cyan, white):
  - the name in minuscule is for color **with bold** (example 'yellow'),
  - the name starting with 'B' is for color **without bold** (example 'Byellow'),
  - the name starting with a **capital** letter is for the background color (example 'Yellow').
- for the special effects (blink, *italic*, **bold**, underline, negative), they might not always be supported, but they usually are:
  - the name in minuscule is used to turn *on* the effect (example 'i' to turn on italic),
  - the name starting in capital letter is used to turn *down* the effect (example 'I' to turn off italic).
- for the other special effects (`nocolors`, `default`, `Default`, `clear`, `el`), the effect is **immediate** (and seems to be well supported).

List of functions

To print a string

- *`sprint()`* (page 12): give a string,
- *`printc()`* (page 12): like `print()`, but with interpreting tags to put colors. **This is the most useful function in this module !**
- *`writec()`* (page 13): like printc, but using any file object (and no new line added at the end of the string).

To clean the terminal or the line

- *erase()* (page 11): erase all ANSI colors tags in the string (like sprint, but erasing and not interpreting color tags),
- *clearLine()* (page 11), *clearScreen()* (page 11): to clear the current line or screen,
- *Reset()* (page 10): to return to default foreground and background, and stopping all *fancy* effects (like blinking or reverse video).

Others functions

- *notify()* (page 11): try to display a *system* notification. **Only on GNU/Linux with notify-send installed.**
- *xtitle()* (page 14): try to set the *title* of the terminal. Warning: **not always supported**.

Example of use (module)

To store a string, use *sprint()* (page 12) (i.e. print to a string, *sprint*), like this:

```
>>> example = sprint("France flag is <blue>blue<white>white<red>red<white>, Italy␣
↪flag have <green>green on it<white>.")
>>> example
'France flag is [01;34mblue[01;37mwhite[01;31mred[01;37m, Italy flag have [01;
↪32mgreen on it[01;37m.'
```

The string `example` can then be printed, with colors, with:

```
>>> print(example)  # Sorry, but in the documentation it is hard to show colors :)
France flag is bluewhitered, Italy flag have green on it.
```

To directly print a string colored by tags, use *printc()* (page 12) (colors will be there if you try this in your terminal):

```
>>> printc("<reset><white>Batman's costum is <black>black<white>, Aquaman's costum is
↪<blue>blue<white> and <green>green<white>.<reset>")
Batman's costum is black, Aquaman's costume is blue and green.
```

See also:

This is the most useful function. To do the same, but on any file, use *writec()* (page 13).

Moreover, the function *erase()* (page 11) can also be useful to simply delete all *valid* color tags:

```
>>> print(erase("<reset>Batman's costum is <black>black<white>, Aquaman's costum is
↪<blue>blue<white> and <green>green<white>, and this is a non-valid <tag>, so it is␣
↪kept like this.<reset>"))
Batman's costum is black, Aquaman's costum is blue and green, and this is a non-valid
↪<tag>, so it is kept like this
```

In this last example, an `<el>` tag (`el`) is used to erase the current content of the line, useful to make a *dynamical* print:

```
>>> writec("<reset><red>Computing <u>len(str(2**562017))<reset>...."); tmp =␣
↪len(str(2**562017)); writec("<el><green>Done !<reset>")
Done !
```

The first part of the line 'Computing len(str(2**562017))....' have disappeared after the computation! (which takes about one second).

Example of use (script)

- To show the help `$ ansicolortags.py --help`;
- To run a test `$ ansicolortags.py --test`;
- To produce a GNU Bash color aliases file `$ ansicolortags.py --generate --file ~/.color_aliases.sh`.

Auto detection

This script can normally detect if ANSI codes are supported :

1. `$ ansicolortags.py --help` : will print with colors if colors seems to be supported;
2. `$ ansicolortags.py --help --noANSI` : will print without any colors, even if it is possible;
3. `$ ansicolortags.py --help --ANSI` : will force the use of colors, even if they seems to be not supported.

And, the module part behaves exactly like the script part.

Elsewhere online

This project can be found on-line:

- here on BitBucket: https://bitbucket.org/lbesson/ansicolortags.py
- here on PyPi: https://pypi.python.org/pypi/ansicolortags

And some documentation on ANSI codes:

- The reference page for ANSI code is : here on Wikipedia.
- A reference page for XTitle escape code is : here.

Copyrigth

Complete documentation

Note: The doc is available on-line, on Read the Docs: http://ansicolortags.readthedocs.io/.

`ansicolortags.`**`Reset`**`()` → unit
Try to reset the current ANSI codes buffer, using `reset`.

ansicolortags.**_generate_color_sh**(*file_name=None*) → string | unit.

Used to print or generate (if file_name is present and is a valid URI address) a profile of all the colors defined in this file.

Print all ANSI Colors as `export NAME="VALUE"`. Useful to automatically generate a `.color.sh` file, to be used with Bash: and now you can easily colorized your Bash script with `. color.sh` to import all colors.

The file is a list of `export NAME="VALUE"`, to be used with GNU Bash.

Note: For example, to generate the `color.sh` file with this script, use the `-g` or `--generate` option, with `-f FILE` or `--file FILE`:

```
$ python -m ansicolortags -g -f color.sh
```

Hint: I suggest to save this `.color.sh` file to your home, like `~/.color.sh`, so it will be available for any GNU Bash script. During the last 4 years, all the Bash scripts I wrote that uses this color profile (or assume it to be enabled, e.g. from your .bashrc file) assume it to be saved as `~/.color.sh`.

For instance, PDFCompress, git-blame-last-commit.sh, mymake.sh, makequotes.sh, photosmagic.sh, remove_trailing_spaces.sh, series.sh, strapdown2pdf.sh, Volume.sh etc.

In a Bash script, I suggest to source this `.color.sh` file like this (it checks if the file exists before sourcing it):

```
[ -f ~/.color.sh ] && . ~/.color.sh
```

---

ansicolortags.**_run_complete_tests**() → unit.

Launch a complete test of all ANSI Colors code in the list `colorList`.

ansicolortags.**clearLine**() → unit

Try to clear the current line using ANSI code `el`.

ansicolortags.**clearScreen**() → unit

Try to clear the screen using ANSI code `clear`.

ansicolortags.**erase**(*chainWithTags*, *left='<'*, *right='>'*, *verbose=False*) → string

Parse a string containing color tags, when color is one of the previous define name, and then return it, with color tags **erased**.

Example:

```
>>> print(erase("<reset><blue>This is blue.<white> And <this> is white.<red> Now␣
↪this is red because I am <angry> !<reset>"))
This is blue. And <this> is white. Now this is red because I am <angry> !
```

This example seems exactly the same that the previous one in the documentation, but it's not (it is impossible to put color in the output of a Python example in Sphinx documentation, so there is **no color in output** in the examples... but be sure there is the real output !).

Warning: This function can mess up a string which has unmatched opening and closing tags (< without a > or > without a <), use it carefully.

ansicolortags.**notify**(*msg=''*, *obj='Notification sent by ansicolortags.notify'*, *icon=None*, *verb=False*) → bool

Notification using `subprocess` and `notify-send` (GNU/Linux command-line program). Also print the

informations directly to the screen (only if verb=True).

> **Warning:** This does not use any *ANSI escape* codes, but the common *notify-send* GNU/Linux command line program. It will probably fail (but cleanly) on Windows or Mac OS X.

- Return True if and only if the title have been correctly changed.
- Fails simply if `notify-send` is not found.

`ansicolortags.`**`printc`**(*chainWithTags*, **objects*, *left='<'*, *right='>'*, *sep=' '*, *end='n'*, *erase=False*, ***kwargs*) → unit

Basically a shortcut to `print(sprint(chainWithTags))` : it analyzes all tags (i.e., it converts the tags like `<red>` to their ANSI code value, like `red`), and then it prints the result.

Example (in a terminal the colors, and the bold and underlining effects would be there):

```
>>> printc("<reset><white>« <u>Fifty shades of <red>red<white><U> » could be a
↪<green>good<white> book, <b>if it existed<B>.<reset>")
« Fifty shades of red » could be a good book, if it existed.
```

It accepts one or more "things" to print, exactly like `print()`: for each value `arg_i` in `*objects`:

- if `arg_i` is a string, it is converted using `sprint(arg_i, left=left, right=right)` (*sprint()* (page 12)), and then passed to `print()`.
- otherwise `arg_i` is passed to `print()` without modification (in the same order, of course).

Example with more than one object:

```
>>> print("OK n =", 17, "and z =", 1 + 5j, ".")
OK n = 17 and z = (1+5j) .
>>> printc("<reset><green>OK<white> n =<magenta>", 17, "<white>and z =<blue>", 1␣
↪+ 5j, "<reset>.")  # in a terminal, the output will have colors:
OK n = 17 and z = (1+5j) .
```

This is the more useful function in this package.

- If `erase = True`, then *erase()* (page 11) is used instead of *sprint()* (page 12)

Hint: I suggest to use ansicolortags.py in your own project with the following piece of code:

```
try:
    from ansicolortags import printc
except ImportError:
    print("WARNING: ansicolortags was not found, disabling colors instead.
↪\nPlease install it with 'pip install ansicolortags'")
    def printc(*a, **kwargs):
        print(*a, **kwargs)
```

Hint: During the last 4 years, a lot of the small Python scripts I wrote try to use this module to add some colors: for example, FreeSMS.py, plotnotes.py, strapdown2html.py, calc_interets.py...

`ansicolortags.`**`sprint`**(*chainWithTags*, *left='<'*, *right='>'*, *verbose=False*) → string

Parse a string containing color tags, when color is one of the previous define name, and then return it, with color tags changed to concrete ANSI color codes.

Tags are delimited by `left` and `right`. By default, it's HTML / Pango style whit '<' and '>', but you can change them.

For example, a custom style even closer to HTML could be: `left='<span color='` and `right = '</span>'` is also possible.

> **Warning:** It is more prudent to put nothing else than ANSI Colors (i.e. values in `colorList`) between `'<'` and `'>'` in `chainWithTags`. The behavior of the function in case of false tags **is not perfect**. Moreover, a good idea could be to try not to use '<' or '>' for anything else than tags. I know, it's not perfect. But, the syntax of color tags is so simple and so beautiful with this limitation that you will surely forgive me this, *won't you* ;) ?

Example (where unknown tags are left unmodified, and the colors should be there):

```
>>> print(sprint("<reset><blue>This is blue.<white> And <this> is white.<red> Now␣
↪this is red because I am <angry> !<green><reset>"))
This is blue. And <this> is white. Now this is red because I am <angry> !
```

This function is used in all the following, so all other function can also use `left` and `right` arguments.

`ansicolortags.`**`tocolor`**(*mystring*) → string
Convert a string to a color. `mystring` **have** to be in `colorDict` to be recognized (and interpreted). Default value if `mystring` is not one of the color name is `""` the empty string.

`ansicolortags.`**`writec`**(*chainWithTags=""*, *out=sys.stdout*, *left='<'*, *right='>'*, *flush=True*) → unit
Useful to print colored text **to a file**, represented by the object `out`. Also useful to print colored text, but without any trailing 'n' character.

In this example, before the long computation begin, it prints 'Computing 2**(2**(2**4))....', and when the computation is done, erases the current line (with `<el>` tag, `el`), and prints ' Done !' in green, and the result of the computation:

```
>>> writec("<red>Computing<reset> 2**(2**(2**4))....."); tmp = 2**(2**(2**4));␣
↪writec("<el><green>Done !<reset>")
Done !
```

This example show how to use this module to write colored data in a file. Be aware that this file now contains ANSI escape sequences. For example, `$ cat /tmp/colored-text.txt` will well print the colors, but editing the file will show *hard values* of escape code:

```
>>> my_file = open('/tmp/colored-text.txt', mode = 'w')  # Open an random file.
>>> write("<reset><blue>this is blue.<white>And <this> is white.<red>Now this is␣
↪red because I am <angry> !<green><reset>", file = my_file)
>>> # Now this file '/tmp/colored-text.txt' has some ANSI colored text in it.
```

Remark: It can also be used to simply reinitialize the ANSI colors buffer, but the function *`Reset()`* (page 10) is here for this:

```
>>> writec("<reset>")
```

> **Warning:** The file `out` **will be flushed** by this function if `flush` is set to `True` (this is default behavior). If you prefer no to, use `flush=False` option:
>
> ```
> >>> writec(chainWithTags_1, out=my_file, flush=False)
> >>> # many things...
> >>> writec(chainWithTags_n, out=my_file, flush=False)
> >>> my_file.flush()  # only flush here!
> ```

ansicolortags.**xtitle**(*new_title=""*, *verb=False*) → 0 or 1

Modify the current terminal title. Returns 0 if one of the two solutions worked, 1 otherwise.

An experimental try is with **ANSI escape code**, if the simple way by calling the `xtitle` program does not work (or if it is not installed).

Note: The second solution simply uses the two *ANSI* Tags `<title>` (`title`) and `<bell>` (`bell`). So, you can also do it with:

```
>>> ansicolortags.writec("<title>This is the new title of the terminal<bell>")
```

But this function *xtitle* is better: it tries two ways, and returns a signal to inform about his success.

TODOs for ansicolortags package ?

1. Improve support of Windows ?
2. Add automated tests.
3. Use travis or another solution to automatically check if the build passes after each commit.
4. Write a comparison with other ANSI color modules, showing the advantages of mine (and disadvantages).
5. Write a page on the doc about the use of `.color.sh` in a GNU bash script.

This project is currently in version 0.4, release public. Last update of this doc was made 09/08/2017, 15h:27m:32s.

Note: This project is based on my old ANSIColors-balises project, which was only for Python 2.7.

The MIT License (MIT)

Cloud of words

Words from ansicolortags.py - (C) 2016 Lilian Besson

Python Module Index

a

ansicolortags, 8

Index

Symbols

_generate_color_sh() (in module ansicolortags), 10
_run_complete_tests() (in module ansicolortags), 11

A

ansicolortags (module), 8

C

clearLine() (in module ansicolortags), 11
clearScreen() (in module ansicolortags), 11

E

erase() (in module ansicolortags), 11

N

notify() (in module ansicolortags), 11

P

printc() (in module ansicolortags), 12

R

Reset() (in module ansicolortags), 10

S

sprint() (in module ansicolortags), 12

T

tocolor() (in module ansicolortags), 13

W

writec() (in module ansicolortags), 13

X

xtitle() (in module ansicolortags), 14