consolekit

Release 1.3.0

Additional utilities for click.

Dominic Davis-Foster

Sep 25, 2021

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install consolekit --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install consolekit
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/consolekit@master --user
```

Additionally, for better support in terminals, install psutil by specifying the `terminals` extra:

```
$ python -m pip install consolekit[terminals]
```

or, if you installed `consolekit` through conda:

```
$ conda install -c conda-forge psutil
```

Chapter

TWO

Highlights

- *options.auto_default_option()*

```
auto_default_option(
  *param_decls,
  **attrs,
  ) -> Callable[[~_C], ~_C]
```

Attaches an option to the command, with a default value determined from the decorated function's signature.

- *input.choice()*

```
choice(
  options: Union[List[str], Mapping[str, str]],
  text: str = '',
  default: Optional[str] = None,
  prompt_suffix: str = ': ',
  show_default: bool = True,
  err: bool = False,
  start_index: int = 0,
  ) -> Union[str, int]
```

Prompts a user for input.

- *commands.MarkdownHelpCommand*

```
MarkdownHelpCommand(
  name: Optional[str],
  context_settings: Optional[Dict[str, Any]] = None,
  callback: Optional[Callable[..., Any]] = None,
  params: Optional[List[Parameter]] = None,
  help: Optional[str] = None,
  epilog: Optional[str] = None,
  short_help: Optional[str] = None,
  options_metavar: Optional[str] = '[OPTIONS]',
  add_help_option: bool = True,
  no_args_is_help: bool = False,
  hidden: bool = False,
  deprecated: bool = False,
  )
```

Subclass of `click.Command` which treats the help text as markdown and prints a rendered representation.

- *commands.SuggestionGroup*

```
SuggestionGroup(
  name: Optional[str] = None,
  commands: Union[Dict[str, Command], Sequence[Command], None] = None,
  **attrs: Any,
  )
```

Subclass of `click.Group` which suggests the most similar command if the command is not found.

Chapter

THREE

consolekit

Additional utilities for click.

Attention: `consolekit` disables Python's readline history to prevent unrelated histories appearing for prompts. If the original behaviour is desired run:

```
import readline
readline.set_history_length(-1)
readline.set_auto_history(True)
```

Functions:

click_command([name, cls])	Shortcut to `click.command()`, with the `-h`/`--help` option enabled and a max width of `120`.
click_group([name, cls])	Shortcut to `click.group()`, with the `-h`/`--help` option enabled and a max width of `120`.
option(*param_decls, **attrs)	Shortcut to `click.option()`, but using *consolekit.input.confirm()* when prompting for a boolean flag.

click_command(*name=None*, *cls=None*, ***attrs*)

Shortcut to `click.command()`, with the `-h`/`--help` option enabled and a max width of `120`.

Parameters

- **name** (`Optional`[`str`]) – Default `None`.
- **cls** (`Type`[`Command`]) – Default `None`.
- ****attrs** – Additional keyword arguments passed to the `Command`.

Return type `Callable`[[`Callable`], `Command`]

click_group(*name=None*, *cls=None*, ***attrs*)

Shortcut to `click.group()`, with the `-h`/`--help` option enabled and a max width of `120`.

Parameters

- **name** (`Optional`[`str`]) – Default `None`.
- **cls** (`Type`[`Group`]) – Default `None`.
- ****attrs** – Additional keyword arguments passed to the `Group`.

Return type `Callable`[[`Callable`], ~_G]

option(**param_decls*, ***attrs*)

Shortcut to `click.option()`, but using *`consolekit.input.confirm()`* when prompting for a boolean flag.

Parameters

- ***param_decls**
- ****attrs** – Additional keyword arguments passed to `click.command()`.

Return type `Callable`[[~_C], ~_C]

Chapter

FOUR

consolekit.commands

Customised click commands and command groups.

New in version 0.8.0.

Classes:

ContextInheritingGroup([name, commands])	Subclass of `click.Group` whose children inherit its `context_settings`.
MarkdownHelpCommand(name[, ...])	Subclass of `click.Command` which treats the help text as markdown and prints a rendered representation.
MarkdownHelpGroup([name, commands])	Subclass of `click.Group` which treats the help text as markdown and prints a rendered representation.
MarkdownHelpMixin()	Mixin class for `click.Command` and `click.Group` which treats the help text as markdown and prints a rendered representation.
RawHelpCommand(name[, context_settings, ...])	Subclass of `click.Command` which leaves the help text unformatted.
RawHelpGroup([name, commands])	Subclass of `click.Group` which leaves the help text unformatted.
RawHelpMixin()	Mixin class for `click.Command` and `click.Group` which leaves the help text unformatted.
SuggestionGroup([name, commands])	Subclass of `click.Group` which suggests the most similar command if the command is not found.

class ContextInheritingGroup(*name=None*, *commands=None*, ***attrs*)

Bases: `Group`

Subclass of `click.Group` whose children inherit its `context_settings`.

The group's commands can be given different context settings by passing the `context_settings` keyword argument to *`command()`* and *`group()`* as normal.

New in version 1.1.0.

Methods:

command(*args, **kwargs)	A shortcut decorator for declaring and attaching a command to the group.
group(*args, **kwargs)	A shortcut decorator for declaring and attaching a group to the group.

command(**args*, ***kwargs*)

A shortcut decorator for declaring and attaching a command to the group.

This takes the same arguments as `click.command()` but immediately registers the created command with this instance by calling into `click.Group.add_command()`.

Return type `Callable`[[`Callable`[..., `Any`]], `Command`]

group(**args*, ***kwargs*)
A shortcut decorator for declaring and attaching a group to the group.

This takes the same arguments as `click.group()` but immediately registers the created group with this instance by calling into `click.Group.add_command()`.

Return type `Callable`[[`Callable`[..., `Any`]], `Group`]

class MarkdownHelpCommand
class MarkdownHelpGroup
Bases: *MarkdownHelpMixin*

Subclasses of `click.Command` and `click.Group` which treat the help text as markdown and print a rendered representation.

Tested in Gnome Terminal and Terminator (both libVTE-based), and PyCharm. libVTE has the best support. PyCharm's support for italics and strikethrough is poor. Support on Windows is, as expected, poor.

Not tested on other terminals, but contributions are welcome to improve support.

New in version 0.8.0.

MarkdownHelpCommand.**parse_args**(*ctx*, *args*)
Parse the given arguments and modify the context as necessary.

Parameters

- **ctx** (`Context`)
- **args** (`List`[`str`])

Return type `List`[`str`]

class MarkdownHelpMixin
Bases: `object`

Mixin class for `click.Command` and `click.Group` which treats the help text as markdown and prints a rendered representation.

See also: *MarkdownHelpCommand* and *MarkdownHelpGroup*

Tip: This can be combined with groups such as *SuggestionGroup*.

Tested in Gnome Terminal and Terminator (both libVTE-based), and PyCharm. libVTE has the best support. PyCharm's support for italics and strikethrough is poor. Support on Windows is, as expected, poor.

Not tested on other terminals, but contributions are welcome to improve support.

New in version 0.8.0.

format_help_text(*ctx*, *formatter*)
Writes the help text to the formatter if it exists.

Parameters

- **ctx** (`Context`)
- **formatter** (`HelpFormatter`)

class RawHelpCommand
class RawHelpGroup
Bases: *RawHelpMixin*

Subclasses of `click.Command` and `click.Group` which leave the help text unformatted.

New in version 0.8.0.

class RawHelpMixin
Bases: `object`

Mixin class for `click.Command` and `click.Group` which leaves the help text unformatted.

See also: *RawHelpCommand* and *RawHelpGroup*

Tip: This can be combined with groups such as *SuggestionGroup*.

New in version 0.8.0.

format_help_text(*ctx*, *formatter*)
Writes the help text to the formatter if it exists.

Parameters

- **ctx** (`Context`)
- **formatter** (`HelpFormatter`)

class SuggestionGroup(*name=None*, *commands=None*, ***attrs*)
Bases: *ContextInheritingGroup*

Subclass of `click.Group` which suggests the most similar command if the command is not found.

Changed in version 0.8.0: Moved to *consolekit.commands*.

Changed in version 1.1.0: Now inherits from *ContextInheritingGroup*

resolve_command(*ctx*, *args*)
Resolve the requested command belonging to this group, and print a suggestion if it can't be found.

Parameters

- **ctx** (`Context`)
- **args** (`List[str]`)

Return type `Tuple[str, Command, List[str]]`

Returns The name of the matching command, the `click.Command` object itself, and any remaining arguments.

Chapter

FIVE

consolekit.input

Input functions (prompt, choice etc.).

Functions:

choice(options[, text, default, ...])	Prompts a user for input.
confirm(text[, default, abort, ...])	Prompts for confirmation (yes/no question).
prompt(text[, default, hide_input, ...])	Prompts a user for input.
stderr_input([prompt, file])	Read a string from standard input, but prompt to standard error.

choice(*options*, *text=''*, *default=None*, *prompt_suffix=': '*, *show_default=True*, *err=False*, *start_index=0*)

Prompts a user for input.

If the user aborts the input by sending an interrupt signal, this function will catch it and raise a `click.Abort` exception.

Parameters

- **options** (`Union[List[str], Mapping[str, str]]`)
- **text** (`str`) – The text to show for the prompt. Default `''`.
- **default** (`Optional[str]`) – The index of the default value to use if the user does not enter anything. If this is not given it will prompt the user until aborted. Default `None`.
- **prompt_suffix** (`str`) – A suffix that should be added to the prompt. Default `': '`.
- **show_default** (`bool`) – Shows or hides the default value in the prompt. Default `True`.
- **err** (`bool`) – If `True` the file defaults to `stderr` instead of `stdout`, the same as with echo. Default `False`.
- **start_index** (`int`) – If `options` is a list of values, sets the start index. Default `0`.

Return type `Union[str, int]`

Overloads

- *choice*(**options**: `List[str]`, **text** = ..., **default** = ..., **prompt_suffix** = ..., **show_default** = ..., **err** = ..., **start_index** = ...) -> `int`
- *choice*(**options**: `Mapping[str, str]`, **text** = ..., **default** = ..., **prompt_suffix** = ..., **show_default** = ..., **err** = ..., **start_index** = ...) -> `str`

confirm(*text*, *default=False*, *abort=False*, *prompt_suffix=': '*, *show_default=True*, *err=False*)

Prompts for confirmation (yes/no question).

If the user aborts the input by sending a interrupt signal this function will catch it and raise a `click.Abort` exception.

Parameters

- **text** (str) – The question to ask.
- **default** (bool) – The default for the prompt. Default False.
- **abort** (bool) – If True a negative answer aborts the exception by raising click.Abort. Default False.
- **prompt_suffix** (str) – A suffix that should be added to the prompt. Default ': '.
- **show_default** (bool) – Shows or hides the default value in the prompt. Default True.
- **err** (bool) – If True the file defaults to stderr instead of stdout, the same as with echo. Default False.

prompt(*text*, *default=None*, *hide_input=False*, *confirmation_prompt=False*, *type=None*, *value_proc=None*, *prompt_suffix=': '*, *show_default=True*, *err=False*, *show_choices=True*)

Prompts a user for input.

If the user aborts the input by sending an interrupt signal, this function will catch it and raise a click.Abort exception.

Parameters

- **text** (str) – The text to show for the prompt.
- **default** (Optional[str]) – The default value to use if no input happens. If this is not given it will prompt until it is aborted. Default None.
- **hide_input** (bool) – If True then the input value will be hidden. Default False.
- **confirmation_prompt** (Union[bool, str]) – Asks for confirmation for the value. Can be set to a string instead of True to customize the message. Default False.
- **type** (Union[type, ParamType, Tuple[Union[type, ParamType], ...], Callable[[str], Any], Callable[[Optional[str]], Any], None]) – The type to check the value against. Default None.
- **value_proc** (Optional[Callable[[Optional[str]], Any]]) – If this parameter is provided it must be a function that is invoked instead of the type conversion to convert a value. Default None.
- **prompt_suffix** (str) – A suffix that should be added to the prompt. Default ': '.
- **show_default** (bool) – Shows or hides the default value in the prompt. Default True.
- **err** (bool) – If True the file defaults to stderr instead of stdout, the same as with click.echo(). Default False.
- **show_choices** (bool) – Show or hide choices if the passed type is a click.Choice. For example, if the choice is either day or week, show_choices is True and text is 'Group by' then the prompt will be 'Group by (day, week): '. Default True.

stderr_input(*prompt=''*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*)

Read a string from standard input, but prompt to standard error.

The trailing newline is stripped. If the user hits EOF (Unix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.

On Unix, GNU readline is used if enabled.

The prompt string, if given, is printed to stderr without a trailing newline before reading.

Return type str

Chapter

SIX

consolekit.options

Command line options.

New in version 0.4.0.

Classes:

DescribedArgument(*args[, description])	`click.Argument` with an additional keyword argument and attribute giving a short description.
MultiValueOption([param_decls, ...])	Subclass of `click.Option` that behaves like argparse's `nargs='+'`.

Data:

_A	Invariant `TypeVar` bound to `click.Argument`.
_C	Invariant `TypeVar` bound to `click.Command`.

Functions:

auto_default_argument(*param_decls, **attrs)	Attaches an argument to the command, with a default value determined from the decorated function's signature.
auto_default_option(*param_decls, **attrs)	Attaches an option to the command, with a default value determined from the decorated function's signature.
colour_option([help_text])	Adds an option (via the parameter `colour: bool`) to enable verbose output.
flag_option(*args[, default])	Decorator to a flag option to a click command.
force_option(help_text)	Decorator to add the `-f / --force` option to a click command.
no_pager_option([help_text])	Decorator to add the `--no-pager` option to a click command.
verbose_option([help_text])	Adds an option (via the parameter `verbose: int`) to enable verbose output.
version_option(callback)	Adds an option to show the version and exit.

class DescribedArgument(**args*, *description=None*, ***kwargs*)

Bases: `Argument`

`click.Argument` with an additional keyword argument and attribute giving a short description.

This is not shown in the help text, but may be useful for manpages or HTML documentation where additional information can be provided.

New in version 1.2.0.

Parameters **description** (Optional[str]) – Default None.

See click.Argument and click.Parameter for descriptions of the other keyword argu ments.

description
Type: Optional[str]

A short description of the argument.

class MultiValueOption(*param_decls=None*, *show_default=False*, *help=None*, *hidden=False*, *type=None*, *required=False*, *default=()*, *callback=None*, *metavar=None*, *expose_value=True*, *is_eager=False*)

Bases: Option

Subclass of click.Option that behaves like argparse's nargs='+'.

New in version 0.6.0.

Parameters

- **param_decls** (Optional[List[str]]) – The parameter declarations for this option or argument. This is a list of flags or argument names. Default None.
- **show_default** (bool) – Controls whether the default value should be shown on the help page. Normally, defaults are not shown. If this value is a string, it shows the string instead of the value. This is particularly useful for dynamic options. Default False.
- **help** (Optional[str]) – The help string. Default None.
- **hidden** (bool) – Hide this option from help outputs. Default False.
- **type** (Union[type, ParamType, Tuple[Union[type, ParamType], ...], Callable[[str], Any], Callable[[Optional[str]], Any], None]) – The type that should be used. Either a click.ParamType or a Python type. The later is converted into the former automatically if supported. Default None.
- **required** (bool) – Controls whether this is optional. Default False.
- **default** (Optional[Any]) – The default value if omitted. This can also be a callable, in which case it is invoked when the default is needed without any arguments. Default ().
- **callback** (Optional[Callable[[Context, Parameter, str], Any]]) – A callback that should be executed after the parameter was matched. This is called as fn(ctx, param, value) and needs to return the value. Default None.
- **metavar** (Optional[str]) – How the value is represented in the help page. Default None.
- **expose_value** (bool) – If True then the value is passed onwards to the command callback and stored on the context, otherwise it is skipped. Default True.
- **is_eager** (bool) – Eager values are processed before non eager ones. Default False.

Example usage:

```
@click.option(
        "--select",
        type=click.STRING,
        help="The checks to enable",
        cls=MultiValueOption,
        )
@click_command()
def main(select: Iterable[str]):
    select = list(select)
```

Methods:

add_to_parser(parser, ctx)	Add the *MultiValueOption* to the given parser.
process_value(ctx, value)	Given a value and context, converts the value as necessary.

add_to_parser(*parser*, *ctx*)
Add the *MultiValueOption* to the given parser.

Parameters

- **parser** (OptionParser)
- **ctx** (Context)

process_value(*ctx*, *value*)
Given a value and context, converts the value as necessary.

Parameters

- **ctx** (Context)
- **value** (Any)

Return type Optional[Tuple]

_A = TypeVar(_A, bound=Argument)
Type: TypeVar

Invariant TypeVar bound to click.Argument.

_C = TypeVar(_C, bound=Command)
Type: TypeVar

Invariant TypeVar bound to click.Command.

auto_default_argument(**param_decls*, ***attrs*)
Attaches an argument to the command, with a default value determined from the decorated function's signature.

All positional arguments are passed as parameter declarations to click.Argument; all keyword arguments are forwarded unchanged (except cls). This is equivalent to creating an click.Argument instance manually and attaching it to the click.Command.params list.

New in version 0.8.0.

Parameters **cls** – the option class to instantiate. This defaults to click.Argument.

Return type Callable[[~_C], ~_C]

auto_default_option(**param_decls*, ***attrs*)
Attaches an option to the command, with a default value determined from the decorated function's signature.

All positional arguments are passed as parameter declarations to click.Option; all keyword arguments are forwarded unchanged (except cls). This is equivalent to creating an click.Option instance manually and attaching it to the click.Command.params list.

New in version 0.7.0.

Parameters **cls** – the option class to instantiate. This defaults to click.Option.

Return type Callable[[~_C], ~_C]

colour_option(*help_text='Whether to use coloured output.'*)
Adds an option (via the parameter `colour: bool`) to enable verbose output.

New in version 0.4.0.

Parameters **help_text** – The help text for the option. Default `'Whether to use coloured output.'`.

Return type `Callable[[~_C], ~_C]`

flag_option(**args*, *default=False*, ***kwargs*)
Decorator to a flag option to a click command.

New in version 0.7.0.

Parameters

- ***args** – Positional arguments passed to `click.option()`.
- **default** (`Optional[bool]`) – The default state of the flag. Default `False`.
- ****kwargs** – Keyword arguments passed to `click.option()`.

Return type `Callable[[~_C], ~_C]`

force_option(*help_text*)
Decorator to add the `-f / --force` option to a click command.

The value is exposed via the parameter `force: bool`.

New in version 0.5.0.

Parameters **help_text** (`str`) – The help text for the option.

Return type `Callable[[~_C], ~_C]`

no_pager_option(*help_text='Disable the output pager.'*)
Decorator to add the `--no-pager` option to a click command.

The value is exposed via the parameter `no_pager: bool`.

New in version 0.5.0.

Parameters **help_text** – The help text for the option. Default `'Disable the output pager.'`.

Return type `Callable[[~_C], ~_C]`

verbose_option(*help_text='Show verbose output.'*)
Adds an option (via the parameter `verbose: int`) to enable verbose output.

The option can be provided multiple times by the user.

New in version 0.4.0.

Parameters **help_text** (`str`) – The help text for the option. Default `'Show verbose output.'`.

Return type `Callable[[~_C], ~_C]`

version_option(*callback*)

Adds an option to show the version and exit.

The option can be provided multiple times by the user. The count is stored as an integer and passed as the third parameter to the callback function.

New in version 0.4.0.

Parameters **callback** (Callable[[Context, Option, int], Any]) – The callback to invoke when the option is provided.

The callback function might look like:

```
def version_callback(ctx: click.Context, param: click.Option, value: int):
    if not value or ctx.resilient_parsing:
        return

    if value > 1:
        click.echo(f"consolekit version {__version__}, Python {sys.version}")
    else:
        click.echo(f"consolekit version {__version__}")

    ctx.exit()
```

Return type Callable[[~_C], ~_C]

Chapter

SEVEN

consolekit.terminal_colours

Functions for adding ANSI character codes to terminal print statements.

See also: http://en.wikipedia.org/wiki/ANSI_escape_code

Classes:

AnsiFore(*args, **kwargs)	ANSI Colour Codes for foreground colour.
AnsiBack(*args, **kwargs)	ANSI Colour Codes for background colour.
AnsiStyle(*args, **kwargs)	ANSI Colour Codes for text style.
AnsiCursor()	Provides methods to control the cursor.
Fore	alias of *consolekit.terminal_colours.AnsiFore*
Back	alias of *consolekit.terminal_colours.AnsiBack*
Style	alias of *consolekit.terminal_colours.AnsiStyle*
AnsiCodes(*args, **kwargs)	Abstract base class for ANSI Codes.
Colour(style, stack, List[str]], reset)	An ANSI escape sequence representing a colour.

Data:

ColourTrilean	Represents the True/False/None state of colour options.

Functions:

resolve_color_default([color])	Helper to get the default value of the color flag.
strip_ansi(value)	Strip ANSI colour codes from the given string to return a plaintext output.

class AnsiFore(**args*, ***kwargs*)

Bases: *AnsiCodes*

ANSI Colour Codes for foreground colour.

The colours can be used as a context manager, a string, or a function.

Valid values are:

- BLACK
- RED
- GREEN
- YELLOW
- BLUE
- MAGENTA
- CYAN
- WHITE
- RESET
- LIGHTBLACK_EX
- LIGHTRED_EX
- LIGHTGREEN_EX
- LIGHTYELLOW_EX
- LIGHTBLUE_EX
- LIGHTMAGENTA_EX
- LIGHTCYAN_EX
- LIGHTWHITE_EX

This class is also available under the shorter alias *Fore*.

class AnsiBack(**args*, ***kwargs*)

Bases: *AnsiCodes*

ANSI Colour Codes for background colour.

The colours can be used as a context manager, a string, or a function.

Valid values are:

- BLACK
- RED
- GREEN
- YELLOW
- BLUE
- MAGENTA
- CYAN
- WHITE
- RESET
- LIGHTBLACK_EX
- LIGHTRED_EX
- LIGHTGREEN_EX
- LIGHTYELLOW_EX
- LIGHTBLUE_EX
- LIGHTMAGENTA_EX
- LIGHTCYAN_EX
- LIGHTWHITE_EX

This class is also available under the shorter alias *Back*.

class AnsiStyle(**args*, ***kwargs*)

Bases: *AnsiCodes*

ANSI Colour Codes for text style.

Valid values are:

- BRIGHT
- DIM
- NORMAL

Additionally, `AnsiStyle.RESET_ALL` can be used to reset the foreground and background colours as well as the text style.

This class is also available under the shorter alias *Style*.

class AnsiCursor

Bases: `object`

Provides methods to control the cursor.

Methods:

UP([n])	Moves the cursor up `n` lines.
DOWN([n])	Moves the cursor down `n` lines.
FORWARD([n])	Moves the cursor forward (right) `n` lines.
BACK([n])	Moves the cursor backward (left) `n` lines.
POS([x, y])	Moves the cursor to the given position.
HIDE()	Hides the cursor.
SHOW()	Shows the cursor.

UP(*n=1*)

Moves the cursor up `n` lines.

Parameters **n** (`int`) – Default `1`.

Return type `str`

DOWN(*n=1*)
Moves the cursor down `n` lines.

Parameters **n** (`int`) – Default `1`.

Return type `str`

FORWARD(*n=1*)
Moves the cursor forward (right) `n` lines.

Parameters **n** (`int`) – Default `1`.

Return type `str`

BACK(*n=1*)
Moves the cursor backward (left) `n` lines.

Parameters **n** (`int`) – Default `1`.

Return type `str`

POS(*x=1*, *y=1*)
Moves the cursor to the given position.

Parameters

- **x** (`int`) – Default `1`.
- **y** (`int`) – Default `1`.

Return type `str`

HIDE()
Hides the cursor.

New in version 0.7.0.

Return type `str`

SHOW()
Shows the cursor.

New in version 0.7.0.

Return type `str`

Fore
alias of *consolekit.terminal_colours.AnsiFore*

Back
alias of *consolekit.terminal_colours.AnsiBack*

Style
alias of *consolekit.terminal_colours.AnsiStyle*

class AnsiCodes(**args*, ***kwargs*)
Bases: `ABC`

Abstract base class for ANSI Codes.

class Colour(*style: str*, *stack: Union[Deque[str], List[str]]*, *reset: str*)

Bases: str

An ANSI escape sequence representing a colour.

The colour can be used as a context manager, a string, or a function.

Parameters

- **style** (str) – Escape sequence representing the style.
- **stack** (List[str]) – The stack to place the escape sequence on.
- **reset** (str) – The escape sequence to reset the style.

Methods:

__call__(text)	Returns the given text in this colour.
from_code(code[, background])	Returns a *Colour* to create coloured text.
from_256_code(code[, background])	Returns a *Colour* to create 256-colour text.
from_rgb(r, g, b[, background])	Returns a *Colour* to create 24-bit coloured text.
from_hex(hex_colour[, background])	Returns a *Colour* to create 24-bit coloured text.

__call__(*text*)

Returns the given text in this colour.

Return type str

classmethod from_code(*code*, *background=False*)

Returns a *Colour* to create coloured text.

The colour can be reset using Fore.RESET or Back.RESET.

New in version 0.9.0.

Parameters

- **code** (Union[str, int]) – A 3- or 4- bit ANSI colour code.
- **background** (bool) – Whether to set the colour for the background. Default False.

Return type *Colour*

Note: The background option only influences the reset value and the stack used. It will not handle conversion of foreground codes to background codes.

classmethod from_256_code(*code*, *background=False*)

Returns a *Colour* to create 256-colour text.

The colour can be reset using Fore.RESET or Back.RESET.

New in version 0.9.0.

Note: Not all terminals support 256-colour mode.

Parameters

- **code** (Union[str, int]) – A 256-colour ANSI code.

- **background** (bool) – Whether to set the colour for the background. Default False.

Return type *Colour*

Note: The background option only influences the reset value and the stack used. It will not handle conversion of foreground codes to background codes.

---

classmethod from_rgb(*r*, *g*, *b*, *background=False*)

Returns a *Colour* to create 24-bit coloured text.

The colour can be reset using Fore.RESET or Back.RESET.

New in version 0.9.0.

Note: Not all terminals support 24-bit colours.

Parameters

- **r** (Union[str, int])
- **g** (Union[str, int])
- **b** (Union[str, int])
- **background** (bool) – Whether to set the colour for the background. Default False.

Return type *Colour*

Note: The background option only influences the reset value and the stack used. It will not handle conversion of foreground codes to background codes.

---

classmethod from_hex(*hex_colour*, *background=False*)

Returns a *Colour* to create 24-bit coloured text.

The colour can be reset using Fore.RESET or Back.RESET.

New in version 0.9.0.

Note: Not all terminals support 24-bit colours.

Parameters

- **hex_colour** (str) – The hex colour code.
- **background** (bool) – Whether to set the colour for the background. Default False.

Return type *Colour*

Note: The background option only influences the reset value and the stack used. It will not handle conversion of foreground codes to background codes.

ColourTrilean
Represents the `True`/`False`/`None` state of colour options.

New in version 0.8.0.

Alias of `Optional[bool]`

resolve_color_default(*color=None*)
Helper to get the default value of the color flag.

If a value is passed it is returned unchanged, otherwise it's looked up from the current context.

If the environment variable `PYCHARM_HOSTED` is `1` (which is the case if running in PyCharm) the output will be coloured by default.

If the environment variable `NO_COLOR` is `1` the output will not be coloured by default. See `https://no-color.org/` for more details. This variable takes precedence over `PYCHARM_HOSTED`.

If no value is passed in, there is no context, and neither environment variable is set, `None` is returned.

Changed in version 1.3.0:

- Added support for the `NO_COLOR` environment variable.
- Only uses a value from the click context (`Context.color`) if it is not `None`. Otherwise falls back to checking the environment variables.

Parameters **color** (`Optional[bool]`) – Default `None`.

Return type `Optional[bool]`

strip_ansi(*value*)
Strip ANSI colour codes from the given string to return a plaintext output.

Parameters **value** (`str`)

Return type `str`

Chapter

EIGHT

consolekit.testing

Test helpers.

New in version 0.9.0.

Attention: This module has the following additional requirements:

```
coincidence>=0.1.0
pytest>=6.0.0
pytest-regressions>=2.0.2
```

These can be installed as follows:

```
$ python -m pip install consolekit[testing]
```

Classes:

CliRunner([charset, env, echo_stdin, mix_stderr])	Provides functionality to invoke and test a Click script in an isolated environment.
Result(runner, stdout_bytes, stderr_bytes, ...)	Holds the captured result of an invoked CLI script.

Functions:

cli_runner()	Returns a click runner for this test function.

class CliRunner(*charset='UTF-8'*, *env=None*, ***, *echo_stdin=False*, *mix_stderr=True*)

Bases: CliRunner

Provides functionality to invoke and test a Click script in an isolated environment.

This only works in single-threaded systems without any concurrency as it changes the global interpreter state.

Parameters

- **charset** (str) – The character set for the input and output data. Default 'UTF-8'.
- **env** (Optional[Mapping[str, str]]) – A dictionary with environment variables to override. Default None.
- **echo_stdin** (bool) – If True, then reading from stdin writes to stdout. This is useful for showing examples in some circumstances. Note that regular prompts will automatically echo the input. Default False.
- **mix_stderr** (bool) – If False, then stdout and stderr are preserved as independent streams. This is useful for Unix-philosophy apps that have predictable stdout and noisy stderr, such that each may be measured independently. Default True.

invoke(*cli*, *args=None*, *input=None*, *env=None*, ***, *catch_exceptions=False*, *color=False*, ***extra*)
Invokes a command in an isolated environment.

The arguments are forwarded directly to the command line script, the `extra` keyword arguments are passed to the `main()` function of the command.

Parameters

- **cli** (`BaseCommand`) – The command to invoke.
- **args** (`Union[str, Iterable[str], None]`) – The arguments to invoke. It may be given as an iterable or a string. When given as string it will be interpreted as a Unix shell command. More details at `shlex.split()`. Default `None`.
- **input** (`Union[bytes, str, IO, None]`) – The input data for `sys.stdin`. Default `None`.
- **env** (`Optional[Mapping[str, str]]`) – The environment overrides. Default `None`.
- **catch_exceptions** (`bool`) – Whether to catch any other exceptions than `SystemExit`. Default `False`.
- **color** (`bool`) – whether the output should contain color codes. The application can still override this explicitly. Default `False`.
- ****extra** – The keyword arguments to pass to `click.Command.main()`.

Return type *Result*

class Result(*runner*, *stdout_bytes*, *stderr_bytes*, *exit_code*, *exception*, *exc_info=None*)
Bases: `Result`

Holds the captured result of an invoked CLI script.

Parameters

- **runner** (`CliRunner`) – The runner that created the result.
- **stdout_bytes** (`bytes`) – The standard output as bytes.
- **stderr_bytes** (`Optional[bytes]`) – The standard error as bytes, or `None` if not available.
- **exit_code** (`int`) – The command's exit code.
- **exception** (`Optional[BaseException]`) – The exception that occurred, if any.
- **exc_info** (`Optional[Tuple[Type[BaseException], BaseException, traceback]]`) – The traceback, if an exception occurred. Default `None`.

Methods:

check_stdout(file_regression[, extension])	Perform a regression check on the standard output from the command.

Attributes:

output	The (standard) output as a string.
stderr	The standard error as a string.
stdout	The standard output as a string.

check_stdout (*file_regression*, *extension='.txt'*, ***kwargs*)
Perform a regression check on the standard output from the command.

Parameters

- **file_regression** (FileRegressionFixture)
- **extension** (str) – The extension of the reference file. Default '.txt'.
- ****kwargs** – Additional keyword arguments passed to FileRegressionFixture.check().

Return type Literal[True]

property output
The (standard) output as a string.

Return type str

property stderr
The standard error as a string.

Return type str

property stdout
The standard output as a string.

Return type str

fixture cli_runner
Scope: function

Returns a click runner for this test function.

Return type *CliRunner*

Chapter

NINE

consolekit.tracebacks

Functions for handling exceptions and their tracebacks.

New in version 1.0.0.

Classes:

TracebackHandler()	Context manager to abort execution with a short error message on the following exception types:

Functions:

handle_tracebacks([show_traceback, cls])	Context manager to conditionally handle tracebacks, usually based on the value of a command line flag.
traceback_handler()	Context manager to abort execution with a short error message on the following exception types:
traceback_option([help_text])	Decorator to add the `-T / --traceback` option to a click command.

class TracebackHandler

Bases: `object`

Context manager to abort execution with a short error message on the following exception types:

- `FileNotFoundError`
- `FileExistsError`

Other custom exception classes inheriting from `Exception` are also handled, but with a generic message.

The following exception classes are ignored:

- `EOFError`
- `KeyboardInterrupt`
- `click.ClickException`
- `SystemExit` (new in version 1.1.2)

How these exceptions are handled can be changed, and supported can be added for further exception classes by subclassing this class. Each method is named in the form `handle_<exception>`, where `exception` is the name of the exception class to handle.

New in version 1.0.0.

See also: *handle_tracebacks()*.

Methods:

__call__()	Use the *TracebackHandler* with a `with` block, and handle any exceptions raised within.
handle(e)	Handle the given exception.

__call__()
Use the *TracebackHandler* with a `with` block, and handle any exceptions raised within.

handle(*e*)
Handle the given exception.

Parameters **e** (`BaseException`)

Return type `bool`

handle_tracebacks(*show_traceback=False*, *cls=<class 'TracebackHandler'>*)
Context manager to conditionally handle tracebacks, usually based on the value of a command line flag.

New in version 0.8.0.

Parameters

- **show_traceback** (`bool`) – If `True`, the full Python traceback will be shown on errors. If `False`, only the summary of the traceback will be shown. In either case the program execution will stop on error. Default `False`.
- **cls** (`Type[TracebackHandler]`) – The class to use to handle the tracebacks. Default *consolekit.tracebacks.TracebackHandler*.

Changed in version 1.0.0: Added the `cls` parameter.

See also: *traceback_handler()* and *TracebackHandler*

Return type `AbstractContextManager[+T_co]`

traceback_handler()
Context manager to abort execution with a short error message on the following exception types:

- `FileNotFoundError`
- `FileExistsError`

Other custom exception classes inheriting from `Exception` are also handled, but with a generic message.

The following exception classes are ignored:

- `EOFError`
- `KeyboardInterrupt`
- `click.ClickException`

New in version 0.8.0.

See also: *handle_tracebacks()* and *TracebackHandler*

traceback_option(*help_text='Show the complete traceback on error.'*)

Decorator to add the `-T / --traceback` option to a click command.

The value is exposed via the parameter `show_traceback: bool`.

New in version 1.0.0.

Parameters **help_text** – The help text for the option. Default `'Show the complete traceback on error.'`.

Return type `Callable[[~_C], ~_C]`

Chapter

TEN

consolekit.utils

Utility functions.

Changed in version 1.0.0: *traceback_handler()* and *handle_tracebacks()* moved to *consolekit.tracebacks*. They will still be importable from here until v2.0.0

Classes:

TerminalRenderer(*extras)	Mistletoe markdown renderer for terminals.

Functions:

abort(message[, colour])	Aborts the program execution.
coloured_diff(a, b[, fromfile, tofile, ...])	Compare two sequences of lines; generate the delta as a unified diff.
hidden_cursor()	Context manager to hide the cursor for the scope of the `with` block.
hide_cursor()	Hide the cursor.
import_commands([source, entry_point])	Returns a list of all commands.
is_command(obj)	Return whether `obj` is a click command.
long_echo(text[, use_pager, colour])	Echo `text` to the terminal, optionally via a pager.
overtype(*objects[, sep, end, file, flush])	Print `objects` to the text stream `file`, starting with `"\r"`, separated by `sep` and followed by `end`.
show_cursor()	Show the cursor.

Data:

braille_spinner	`itertools.cycle()` of braille characters to use as a loading spinner.
snake_spinner	`itertools.cycle()` of braille characters to use as a loading spinner which looks like a snake.
solidus_spinner	`itertools.cycle()` of characters to use as a loading spinner.

class TerminalRenderer(**extras*)

Bases: BaseRenderer

Mistletoe markdown renderer for terminals.

Tested in Gnome Terminal and Terminator (both libVTE-based), and PyCharm. libVTE has the best support. PyCharm's support for italics and strikethrough is poor. Support on Windows is, as expected, poor.

Not tested on other terminals, but contributions are welcome to improve support.

New in version 0.8.0.

Methods:

render(token)	Render the given token for display in a terminal.
render_emphasis(token)	Render emphasis (`*emphasis*`).
render_inline_code(token)	Render inline code (`` `code` ``).
render_line_break(token)	Render a line break in a multiline paragraph.
render_list(token)	Render a markdown list.
render_list_item(token)	Render a markdown list item.
render_paragraph(token)	Render a paragraph.
render_strikethrough(token)	Render strikethrough (`~~strikethrough~~`).
render_strong(token)	Render strong (`**strong**`).

render(*token*)

Render the given token for display in a terminal.

Parameters **token**

Return type str

render_emphasis(*token*)

Render emphasis (`*emphasis*`).

Parameters **token** (Emphasis) – The token to render.

Return type str

render_inline_code(*token*)

Render inline code (`` `code` ``).

Parameters **token** (InlineCode) – The token to render.

Return type str

static render_line_break(*token*)

Render a line break in a multiline paragraph.

Parameters **token**

render_list(*token*)

Render a markdown list.

Parameters **token** (List) – The token to render.

Return type str

render_list_item(*token*)
Render a markdown list item.

Parameters **token** (ListItem)

Return type str

render_paragraph(*token*)
Render a paragraph.

Parameters **token** (Paragraph) – The token to render.

Return type str

render_strikethrough(*token*)
Render strikethrough (~~strikethrough~~).

Parameters **token** (Strikethrough) – The token to render.

Return type str

render_strong(*token*)
Render strong (**strong**).

Parameters **token** (Strong) – The token to render.

Return type str

abort(*message*, *colour=None*)
Aborts the program execution.

Parameters

- **message** (str)
- **colour** (Optional[bool]) – Whether to use coloured output. Default auto-detect.

Changed in version 1.0.1: Added the colour option.

Return type Exception

braille_spinner = <itertools.cycle object>
Type: cycle()

itertools.cycle() of braille characters to use as a loading spinner.

New in version 0.7.0.

coloured_diff(*a*, *b*, *fromfile=''*, *tofile=''*, *fromfiledate=''*, *tofiledate=''*, *n=3*, *lineterm='\n'*, *removed_colour='\x1b[31m'*, *added_colour='\x1b[32m'*)
Compare two sequences of lines; generate the delta as a unified diff.

Unified diffs are a compact way of showing line changes and a few lines of context. The number of context lines is set by n which defaults to three.

By default, the diff control lines (those with ---, +++, or @@) are created with a trailing newline. This is helpful so that inputs created from file.readlines() result in diffs that are suitable for file.writelines() since both the inputs and outputs have trailing newlines.

For inputs that do not have trailing newlines, set the lineterm argument to '' so that the output will be uniformly newline free.

The unidiff format normally has a header for filenames and modification times. Any or all of these may be specified using strings for `fromfile`, `tofile`, `fromfiledate`, and `tofiledate`. The modification times are normally expressed in the ISO 8601 format.

New in version 0.3.0.

Example:

```
>>> for line in coloured_diff(
...     'one two three four'.split(),
...     'zero one tree four'.split(), 'Original', 'Current',
...     '2005-01-26 23:30:50', '2010-04-02 10:20:52',
...     lineterm='',
...     ):
...     print(line)
--- Original        2005-01-26 23:30:50
+++ Current         2010-04-02 10:20:52
@@ -1,4 +1,4 @@
+zero
one
-two
-three
+tree
four
```

Parameters

- **a** (Sequence[str])
- **b** (Sequence[str])
- **fromfile** (str) – Default ''.
- **tofile** (str) – Default ''.
- **fromfiledate** (str) – Default ''.
- **tofiledate** (str) – Default ''.
- **n** (int) – Default 3.
- **lineterm** (str) – Default '\n'.
- **removed_colour** (*Colour*) – The *Colour* to use for lines that were removed. Default '\x1b[31m'.
- **added_colour** (*Colour*) – The *Colour* to use for lines that were added. Default '\x1b[32m'.

Return type str

hidden_cursor()

Context manager to hide the cursor for the scope of the with block.

New in version 0.7.0.

Changed in version 0.9.0: Moved to *consolekit.utils*.

Return type Iterator

hide_cursor()

Hide the cursor.

To show it again use *show_cursor()*, or use the *hidden_cursor()* context manager.

New in version 0.7.0.

import_commands(*source=None*, *entry_point=None*)
Returns a list of all commands.

Commands can be defined locally in the module given in `source`, or by third party extensions who define an entry point in the following format:

```
<name (can be anything)> = <module name>:<command>
```

Parameters

- **source** (Optional[ModuleType]) – Default None.
- **entry_point** (Optional[str]) – Default None.

Return type List[Command]

is_command(*obj*)
Return whether `obj` is a click command.

Parameters **obj**

Return type bool

long_echo(*text*, *use_pager=None*, *colour=None*)
Echo `text` to the terminal, optionally via a pager.

New in version 1.2.0.

Parameters

- **text** (Union[str, StringList, Iterable[str]])
- **use_pager** (Optional[bool]) – If True, forces the use of the pager. If False the pager is never used. If None the pager is used if *sys.stdout`* is a TTY and the number of lines is less than the terminal height. Default None.
- **colour** (Optional[bool]) – Whether to use coloured output. Default auto-detect.

Tip: Allow the user to control the value of `use_pager` with the `no_pager_option()` decorator.

overtype(**objects*, *sep=' '*, *end=''*, *file=None*, *flush=False*)
Print `objects` to the text stream `file`, starting with `"\r"`, separated by `sep` and followed by `end`.

`sep`, `end`, `file` and `flush`, if present, must be given as keyword arguments

All non-keyword arguments are converted to strings like `str` does and written to the stream, separated by *sep* and followed by *end*. If no such arguments are given, *overtype()* will just write `"\r"`.

Parameters

- **objects** – A list of strings or string-like objects to write to the terminal.
- **sep** (str) – String to separate the objects with. Default `'␣'`.
- **end** (str) – String to end with. Default `''`.
- **file** (Optional[IO]) – An object with a `write(string)` method. Default `sys.stdout`.
- **flush** (bool) – If True, the stream is forcibly flushed. Default False.

show_cursor()
Show the cursor.

See also: The *hidden_cursor()* context manager.

New in version 0.7.0.

snake_spinner = <itertools.cycle object>
Type: cycle()

itertools.cycle() of braille characters to use as a loading spinner which looks like a snake.

New in version 1.1.0.

solidus_spinner = <itertools.cycle object>
Type: cycle()

itertools.cycle() of characters to use as a loading spinner.

New in version 0.7.0.

Python Module Index

C

Index

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V