vistir Documentation

Release 0.8.0

Dan Ryan <dan@danryan.co>

Mar 03, 2023

Contents:

CHAPTER 1

vistir: Setup / utilities which most projects eventually need

1.1 Installation

Install from PyPI:

```
$ pipenv install vistir
```

Install from Github:

```
$ pipenv install -e git+https://github.com/sarugaku/vistir.git#egg=vistir
```

1.2 Summary

vistir is a library full of utility functions designed to make life easier. Here are some of the places where these functions are used:

- pipenv
- requirementslib
- pip-tools
- passa

- pythonfinder

1.3 Usage

1.3.1 Importing a utility

You can import utilities directly from **vistir**:

```
from vistir import cd
cd('/path/to/somedir'):
    do_stuff_in('somedir')
```

1.4 Functionality

vistir provides several categories of functionality, including:

- Backports
- Compatibility Shims
- Context Managers
- Miscellaneous Utilities
- Path Utilities

Note: The backports should be imported via `compat` which will provide the native versions of the backported items if possible.

1.4.1 Context Managers

vistir provides the following context managers as utility contexts:

- *atomic_open_for_write()*
- *cd()*
- *open_file()*
- *replaced_stream()*
- *replaced_streams()*
- spinner()
- *temp_environ()*
- *temp_path()*

1.4.1.1 atomic_open_for_write

This context manager ensures that a file only gets overwritten if the contents can be successfully written in its place. If you open a file for writing and then fail in the middle under normal circumstances, your original file is already gone.

```
>>> fn = "test_file.txt"
>>> with open(fn, "w") as fh:
        fh.write("this is some test text")
>>> read_test_file()
this is some test text
>>> def raise_exception_while_writing(filename):
        with vistir.contextmanagers.atomic_open_for_write(filename) as fh:
            fh.write("Overwriting all the text from before with even newer text")
            raise RuntimeError("But did it get overwritten now?")
>>> raise_exception_while_writing(fn)
    Traceback (most recent call last):
        ...
    RuntimeError: But did it get overwritten now?
>>> read_test_file()
    writing some new text
```

1.4.1.2 cd

A context manager for temporarily changing the working directory.

```
>>> os.path.abspath(os.curdir)
'/tmp/test'
>>> with vistir.contextmanagers.cd('/tmp/vistir_test'):
        print(os.path.abspath(os.curdir))
/tmp/vistir_test
```

1.4.1.3 open_file

A context manager for streaming file contents, either local or remote. It is recommended to pair this with an iterator which employs a sensible chunk size.

```
>>> filecontents = b""
    with vistir.contextmanagers.open_file("https://norvig.com/big.txt") as fp:
        for chunk in iter(lambda: fp.read(16384), b""):
            filecontents.append(chunk)
>>> import io
>>> import shutil
>>> filecontents = io.BytesIO(b"")
>>> with vistir.contextmanagers.open_file("https://norvig.com/big.txt") as fp:
        shutil.copyfileobj(fp, filecontents)
```

1.4.1.4 replaced_stream

A context manager to temporarily swap out *stream_name* with a stream wrapper. This will capture the stream output and prevent it from being written as normal.

```
>>> orig_stdout = sys.stdout
>>> with replaced_stream("stdout") as stdout:
...     sys.stdout.write("hello")
...     assert stdout.getvalue() == "hello"

>>> sys.stdout.write("hello")
'hello'
```

1.4.1.5 replaced_streams

Temporarily replaces both *sys.stdout* and *sys.stderr* and captures anything written to these respective targets.

```
>>> import sys
>>> with vistir.contextmanagers.replaced_streams() as streams:
>>>     stdout, stderr = streams
>>>     sys.stderr.write("test")
>>>     sys.stdout.write("hello")
>>>     assert stdout.getvalue() == "hello"
>>>     assert stderr.getvalue() == "test"

>>> stdout.getvalue()
'hello'

>>> stderr.getvalue()
'test'
```

1.4.1.6 spinner

A context manager for wrapping some actions with a threaded, interrupt-safe spinner. The spinner is fully compatible with all terminals (you can use `bouncingBar` on non-utf8 terminals) and will allow you to update the text of the spinner itself by simply setting `spinner.text` or write lines to the screen above the spinner by using `spinner.write(line)`. Success text can be indicated using `spinner.ok("Text")` and failure text can be indicated with `spinner.fail("Fail text")`.

```
>>> lines = ["a", "b"]
>>> with vistir.contextmanagers.spinner(spinner_name="dots", text="Running...",␣
↪handler_map={}, nospin=False) as sp:
        for line in lines:
        sp.write(line + "\n")
        while some_variable = some_queue.pop():
            sp.text = "Consuming item: %s" % some_variable
        if success_condition:
            sp.ok("Succeeded!")
        else:
            sp.fail("Failed!")
```

1.4.1.7 temp_environ

Sets a temporary environment context to freely manipulate `os.environ` which will be reset upon exiting the context.

```
>>> os.environ['MY_KEY'] = "test"
>>> os.environ['MY_KEY']
```

(continues on next page)

(continued from previous page)

```
'test'
>>> with vistir.contextmanagers.temp_environ():
        os.environ['MY_KEY'] = "another thing"
        print("New key: %s" % os.environ['MY_KEY'])
New key: another thing
>>> os.environ['MY_KEY']
'test'
```

1.4.1.8 temp_path

Sets a temporary environment context to freely manipulate `sys.path` which will be reset upon exiting the context.

```
>>> path_from_virtualenv = load_path("/path/to/venv/bin/python")
>>> print(sys.path)
['/home/user/.pyenv/versions/3.7.0/bin', '/home/user/.pyenv/versions/3.7.0/lib/
↪python37.zip', '/home/user/.pyenv/versions/3.7.0/lib/python3.7', '/home/user/.pyenv/
↪versions/3.7.0/lib/python3.7/lib-dynload', '/home/user/.pyenv/versions/3.7.0/lib/
↪python3.7/site-packages']
>>> with temp_path():
        sys.path = path_from_virtualenv
        # Running in the context of the path above
        run(["pip", "install", "stuff"])
>>> print(sys.path)
['/home/user/.pyenv/versions/3.7.0/bin', '/home/user/.pyenv/versions/3.7.0/lib/
↪python37.zip', '/home/user/.pyenv/versions/3.7.0/lib/python3.7', '/home/user/.pyenv/
↪versions/3.7.0/lib/python3.7/lib-dynload', '/home/user/.pyenv/versions/3.7.0/lib/
↪python3.7/site-packages']
```

1.4.2 Miscellaneous Utilities

The following Miscellaneous utilities are available as helper methods:

- *shell_escape()*
- *unnest()*
- dedup()
- *run()*
- *load_path()*
- *partialclass()*
- *to_text()*
- *to_bytes()*
- divide()
- take()
- chunked()
- *decode_for_output()*
- *get_canonical_encoding_name()*
- *get_wrapped_stream()*

- *StreamWrapper*
- get_text_stream()
- replace_with_text_stream()
- get_text_stdin()
- get_text_stdout()
- get_text_stderr()
- echo()

1.4.2.1 shell_escape

Escapes a string for use as shell input when passing *shell=True* to os.Popen().

```
>>> vistir.misc.shell_escape("/tmp/test/test script.py hello")
'/tmp/test/test script.py hello'
```

1.4.2.2 unnest

Unnests nested iterables into a flattened one.

```
>>> nested_iterable = (1234, (3456, 4398345, (234234)), (2396, (23895750, 9283798,␣
↪29384, (289375983275, 293759, 2347, (2098, 7987, 27599)))))
>>> list(vistir.misc.unnest(nested_iterable))
[1234, 3456, 4398345, 234234, 2396, 23895750, 9283798, 29384, 289375983275, 293759,␣
↪2347, 2098, 7987, 27599]
```

1.4.2.3 dedup

Deduplicates an iterable (like a set, but preserving order).

```
>>> iterable = ["repeatedval", "uniqueval", "repeatedval", "anotherval",
↪"somethingelse"]
>>> list(vistir.misc.dedup(iterable))
['repeatedval', 'uniqueval', 'anotherval', 'somethingelse']
```

1.4.2.4 run

Runs the given command using subprocess.Popen() and passing sane defaults.

```
>>> out, err = vistir.run(["cat", "/proc/version"])
>>> out
'Linux version 4.15.0-27-generic (buildd@lgw01-amd64-044) (gcc version 7.3.0 (Ubuntu␣
↪7.3.0-16ubuntu3)) #29-Ubuntu SMP Wed Jul 11 08:21:57 UTC 2018'
```

1.4.2.5 load_path

Load the sys.path from the given python executable's environment as json.

```
>>> load_path("/home/user/.virtualenvs/requirementslib-5MhGuG3C/bin/python")
['', '/home/user/.virtualenvs/requirementslib-5MhGuG3C/lib/python37.zip', '/home/user/
↪.virtualenvs/requirementslib-5MhGuG3C/lib/python3.7', '/home/user/.virtualenvs/
↪requirementslib-5MhGuG3C/lib/python3.7/lib-dynload', '/home/user/.pyenv/versions/3.
↪7.0/lib/python3.7', '/home/user/.virtualenvs/requirementslib-5MhGuG3C/lib/python3.7/
↪site-packages', '/home/user/git/requirementslib/src']
```

1.4.2.6 partialclass

Create a partially instantiated class.

```
>>> source = partialclass(Source, url="https://pypi.org/simple")
>>> new_source = source(name="pypi")
>>> new_source
<__main__.Source object at 0x7f23af189b38>
>>> new_source.__dict__
{'url': 'https://pypi.org/simple', 'verify_ssl': True, 'name': 'pypi'}
```

1.4.2.7 to_text

Convert arbitrary text-formattable input to text while handling errors.

```
>>> vistir.misc.to_text(b"these are bytes")
'these are bytes'
```

1.4.2.8 to_bytes

Converts arbitrary byte-convertable input to bytes while handling errors.

```
>>> vistir.misc.to_bytes("this is some text")
b'this is some text'
>>> vistir.misc.to_bytes(u"this is some text")
b'this is some text'
```

1.4.2.9 chunked

Splits an iterable up into groups *of the specified length*, per more itertools. Returns an iterable.

This example will create groups of chunk size **5**, which means there will be *6 groups*.

```
>>> chunked_iterable = vistir.misc.chunked(5, range(30))
>>> for chunk in chunked_iterable:
...     add_to_some_queue(chunk)
```

1.4.2.10 take

Take elements from the supplied iterable without consuming it.

```
>>> iterable = range(30)
>>> first_10 = take(10, iterable)
>>> [i for i in first_10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> [i for i in iterable]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
↪ 24, 25, 26, 27, 28, 29]
```

1.4.2.11 divide

Splits an iterable up into the *specified number of groups*, per more itertools. Returns an iterable.

```
>>> iterable = range(30)
>>> groups = []
>>> for grp in vistir.misc.divide(3, iterable):
...     groups.append(grp)
>>> groups
[<tuple_iterator object at 0x7fb7966006a0>, <tuple_iterator object at 0x7fb796652780>,
↪ <tuple_iterator object at 0x7fb79650a2b0>]
```

1.4.2.12 decode_for_output

Converts an arbitrary text input to output which is encoded for printing to terminal outputs using the system preferred locale using `locale.getpreferredencoding(False)` with some additional hackery on linux systems.

```
>>> vistir.misc.decode_for_output(u"Some text")
"some default locale encoded text"
```

1.4.2.13 get_canonical_encoding_name

Given an encoding name, get the canonical name from a codec lookup.

```
>>> vistir.misc.get_canonical_encoding_name("utf8")
"utf-8"
```

1.4.2.14 get_wrapped_stream

Given a stream, wrap it in a *StreamWrapper* instance and return the wrapped stream.

```
>>> stream = sys.stdout
>>> wrapped_stream = vistir.misc.get_wrapped_stream(sys.stdout)
```

1.4.2.15 StreamWrapper

A stream wrapper and compatibility class for handling wrapping file-like stream objects which may be used in place of `sys.stdout` and other streams.

```
>>> wrapped_stream = vistir.misc.StreamWrapper(sys.stdout, encoding="utf-8", errors=
↪"replace", line_buffering=True)
>>> wrapped_stream = vistir.misc.StreamWrapper(io.StringIO(), encoding="utf-8",␣
↪errors="replace", line_buffering=True)
```

1.4.2.16 get_text_stream

An implementation of the **StreamWrapper** for the purpose of wrapping **sys.stdin** or **sys.stdout**.

On Windows, this returns the appropriate handle to the requested output stream.

```
>>> text_stream = vistir.misc.get_text_stream("stdout")
>>> sys.stdout = text_stream
>>> sys.stdin = vistir.misc.get_text_stream("stdin")
>>> vistir.misc.echo(u"\04499", fg="green")
```

1.4.2.17 replace_with_text_stream

Given a text stream name, replaces the text stream with a **StreamWrapper** instance.

```
>>> vistir.misc.replace_with_text_stream("stdout")
```

Once invoked, the standard stream in question is replaced with the required wrapper, turning it into a `TextIOWrapper` compatible stream (which ensures that unicode characters can be written to it).

1.4.2.18 get_text_stdin

A helper function for calling **get_text_stream("stdin")**.

1.4.2.19 get_text_stdout

A helper function for calling **get_text_stream("stdout")**.

1.4.2.20 get_text_stderr

A helper function for calling **get_text_stream("stderr")**.

1.4.2.21 echo

Writes colored, stream-compatible output to the desired handle (`sys.stdout` by default).

```
>>> vistir.misc.echo("some text", fg="green", bg="black", style="bold", err=True)  #␣
↪write to stderr
some text
>>> vistir.misc.echo("some other text", fg="cyan", bg="white", style="underline")  #␣
↪write to stdout
some other text
```

1.4.3 Path Utilities

vistir provides utilities for interacting with filesystem paths:

- *vistir.path.normalize_path()*
- *vistir.path.is_in_path()*
- *vistir.path.get_converted_relative_path()*
- *vistir.path.handle_remove_readonly()*
- *vistir.path.is_file_url()*
- *vistir.path.is_readonly_path()*
- *vistir.path.is_valid_url()*
- *vistir.path.mkdir_p()*
- *vistir.path.ensure_mkdir_p()*
- *vistir.path.create_tracked_tempdir()*
- *vistir.path.create_tracked_tempfile()*
- *vistir.path.path_to_url()*
- *vistir.path.rmtree()*
- *vistir.path.safe_expandvars()*
- *vistir.path.set_write_bit()*
- *vistir.path.url_to_path()*
- *vistir.path.walk_up()*

1.4.3.1 normalize_path

Return a case-normalized absolute variable-expanded path.

```
>>> vistir.path.normalize_path("~/${USER}")
/home/user/user
```

1.4.3.2 is_in_path

Determine if the provided full path is in the given parent root.

```
>>> vistir.path.is_in_path("~/.pyenv/versions/3.7.1/bin/python", "${PYENV_ROOT}/
↪versions")
True
```

1.4.3.3 get_converted_relative_path

Convert the supplied path to a relative path (relative to `os.curdir`)

```
>>> os.chdir('/home/user/code/myrepo/myfolder')
>>> vistir.path.get_converted_relative_path('/home/user/code/file.zip')
'./../../file.zip'
>>> vistir.path.get_converted_relative_path('/home/user/code/myrepo/myfolder/
↪mysubfolder')
'./mysubfolder'
>>> vistir.path.get_converted_relative_path('/home/user/code/myrepo/myfolder')
'.'
```

1.4.3.4 handle_remove_readonly

Error handler for shutil.rmtree.

Windows source repo folders are read-only by default, so this error handler attempts to set them as writeable and then proceed with deletion.

This function will call check *vistir.path.is_readonly_path()* before attempting to call *vistir.path.set_write_bit()* on the target path and try again.

1.4.3.5 is_file_url

Checks whether the given url is a properly formatted `file://` uri.

```
>>> vistir.path.is_file_url('file:///home/user/somefile.zip')
True
>>> vistir.path.is_file_url('/home/user/somefile.zip')
False
```

1.4.3.6 is_readonly_path

Check if a provided path exists and is readonly by checking for `bool(path.stat & stat.S_IREAD) and not os.access(path, os.W_OK)`

```
>>> vistir.path.is_readonly_path('/etc/passwd')
True
>>> vistir.path.is_readonly_path('/home/user/.bashrc')
False
```

1.4.3.7 is_valid_url

Checks whether a URL is valid and parseable by checking for the presence of a scheme and a netloc.

```
>>> vistir.path.is_valid_url("https://google.com")
True
>>> vistir.path.is_valid_url("/home/user/somefile")
False
```

1.4.3.8 mkdir_p

Recursively creates the target directory and all of its parents if they do not already exist. Fails silently if they do.

```
>>> os.mkdir('/tmp/test_dir')
>>> os.listdir('/tmp/test_dir')
[]
>>> vistir.path.mkdir_p('/tmp/test_dir/child/subchild/subsubchild')
>>> os.listdir('/tmp/test_dir/child/subchild')
['subsubchild']
```

1.4.3.9 ensure_mkdir_p

A decorator which ensures that the caller function's return value is created as a directory on the filesystem.

```
>>> @ensure_mkdir_p
def return_fake_value(path):
    return path
>>> return_fake_value('/tmp/test_dir')
>>> os.listdir('/tmp/test_dir')
[]
>>> return_fake_value('/tmp/test_dir/child/subchild/subsubchild')
>>> os.listdir('/tmp/test_dir/child/subchild')
['subsubchild']
```

1.4.3.10 create_tracked_tempdir

Creates a tracked temporary directory using `TemporaryDirectory`, but does not remove the directory when the return value goes out of scope, instead registers a handler to cleanup on program exit.

```
>>> temp_dir = vistir.path.create_tracked_tempdir(prefix="test_dir")
>>> assert temp_dir.startswith("test_dir")
True
>>> with vistir.path.create_tracked_tempdir(prefix="test_dir") as temp_dir:
    with io.open(os.path.join(temp_dir, "test_file.txt"), "w") as fh:
        fh.write("this is a test")
>>> os.listdir(temp_dir)
```

1.4.3.11 create_tracked_tempfile

Creates a tracked temporary file using `vistir.compat.NamedTemporaryFile`, but creates a `weakref.finalize` call which will detach on garbage collection to close and delete the file.

```
>>> temp_file = vistir.path.create_tracked_tempfile(prefix="requirements", suffix="txt
↪")
>>> temp_file.write("some\nstuff")
>>> exit()
```

1.4.3.12 path_to_url

Convert the supplied local path to a file uri.

```
>>> path_to_url("/home/user/code/myrepo/myfile.zip")
'file:///home/user/code/myrepo/myfile.zip'
```

1.4.3.13 rmtree

Stand-in for `rmtree()` with additional error-handling.

This version of *rmtree* handles read-only paths, especially in the case of index files written by certain source control systems.

```
>>> vistir.path.rmtree('/tmp/test_dir')
>>> [d for d in os.listdir('/tmp') if 'test_dir' in d]
[]
```

Note: Setting *ignore_errors=True* may cause this to silently fail to delete the path

1.4.3.14 safe_expandvars

Call `os.path.expandvars()` if value is a string, otherwise do nothing.

```
>>> os.environ['TEST_VAR'] = "MY_TEST_VALUE"
>>> vistir.path.safe_expandvars("https://myuser:${TEST_VAR}@myfakewebsite.com")
'https://myuser:MY_TEST_VALUE@myfakewebsite.com'
```

1.4.3.15 set_write_bit

Set read-write permissions for the current user on the target path. Fail silently if the path doesn't exist.

```
>>> vistir.path.set_write_bit('/path/to/some/file')
>>> with open('/path/to/some/file', 'w') as fh:
        fh.write("test text!")
```

1.4.3.16 url_to_path

Convert a valid file url to a local filesystem path. Follows logic taken from pip.

```
>>> vistir.path.url_to_path("file:///home/user/somefile.zip")
'/home/user/somefile.zip'
```

CHAPTER 2

vistir package

2.1 Submodules

2.1.1 vistir.cmdparse module

exception vistir.cmdparse.**ScriptEmptyError**
Bases: ValueError

class vistir.cmdparse.**Script**(*command*, *args=None*)
Bases: object

Parse a script line (in Pipfile's [scripts] section).

This always works in POSIX mode, even on Windows.

args

cmdify()
Encode into a cmd-executable string.

This re-implements CreateProcess's quoting logic to turn a list of arguments into one single string for the shell to interpret.

- All double quotes are escaped with a backslash.
- Existing backslashes before a quote are doubled, so they are all escaped properly.
- Backslashes elsewhere are left as-is; cmd will interpret them literally.

The result is then quoted into a pair of double quotes to be grouped.

An argument is intentionally not quoted if it does not contain whitespaces. This is done to be compatible with Windows built-in commands that don't work well with quotes, e.g. everything with *echo*, and DOS-style (forward slash) switches.

The intended use of this function is to pre-process an argument list before passing it into `subprocess.Popen(..., shell=True)`.

See also: https://docs.python.org/3/library/subprocess.html#converting-argument-sequence

command

extend(*extra_args*)

classmethod parse(*value*)

2.1.2 vistir.compat module

2.1.3 vistir.contextmanagers module

vistir.contextmanagers.**temp_environ**()
Allow the ability to set os.environ temporarily

vistir.contextmanagers.**temp_path**()
A context manager which allows the ability to set sys.path temporarily

```
>>> path_from_virtualenv = load_path("/path/to/venv/bin/python")
>>> print(sys.path)
[
    '/home/user/.pyenv/versions/3.7.0/bin',
    '/home/user/.pyenv/versions/3.7.0/lib/python37.zip',
    '/home/user/.pyenv/versions/3.7.0/lib/python3.7',
    '/home/user/.pyenv/versions/3.7.0/lib/python3.7/lib-dynload',
    '/home/user/.pyenv/versions/3.7.0/lib/python3.7/site-packages'
]
>>> with temp_path():
        sys.path = path_from_virtualenv
        # Running in the context of the path above
        run(["pip", "install", "stuff"])
>>> print(sys.path)
[
    '/home/user/.pyenv/versions/3.7.0/bin',
    '/home/user/.pyenv/versions/3.7.0/lib/python37.zip',
    '/home/user/.pyenv/versions/3.7.0/lib/python3.7',
    '/home/user/.pyenv/versions/3.7.0/lib/python3.7/lib-dynload',
    '/home/user/.pyenv/versions/3.7.0/lib/python3.7/site-packages'
]
```

vistir.contextmanagers.**cd**(*path*)
Context manager to temporarily change working directories

Parameters **path** (*str*) – The directory to move into

```
>>> print(os.path.abspath(os.curdir))
'/home/user/code/myrepo'
>>> with cd("/home/user/code/otherdir/subdir"):
...     print("Changed directory: %s" % os.path.abspath(os.curdir))
Changed directory: /home/user/code/otherdir/subdir
>>> print(os.path.abspath(os.curdir))
'/home/user/code/myrepo'
```

vistir.contextmanagers.**atomic_open_for_write**(*target*, *binary=False*, *newline=None*, *encoding=None*)
Atomically open *target* for writing.

This is based on Lektor's *atomic_open()* utility, but simplified a lot to handle only writing, and skip many multi-process/thread edge cases handled by Werkzeug.

Parameters

- **target** (*str*) – Target filename to write
- **binary** (*bool*) – Whether to open in binary mode, default False
- **newline** (*Optional[str]*) – The newline character to use when writing, determined from system if not supplied.
- **encoding** (*Optional[str]*) – The encoding to use when writing, defaults to system encoding.

How this works:

- Create a temp file (in the same directory of the actual target), and yield for surrounding code to write to it.
- If some thing goes wrong, try to remove the temp file. The actual target is not touched whatsoever.
- If everything goes well, close the temp file, and replace the actual target with this new file.

```
>>> fn = "test_file.txt"
>>> def read_test_file(filename=fn):
        with open(filename, 'r') as fh:
            print(fh.read().strip())

>>> with open(fn, "w") as fh:
        fh.write("this is some test text")
>>> read_test_file()
this is some test text

>>> def raise_exception_while_writing(filename):
        with open(filename, "w") as fh:
            fh.write("writing some new text")
            raise RuntimeError("Uh oh, hope your file didn't get overwritten")

>>> raise_exception_while_writing(fn)
Traceback (most recent call last):
    ...
RuntimeError: Uh oh, hope your file didn't get overwritten
>>> read_test_file()
writing some new text

# Now try with vistir
>>> def raise_exception_while_writing(filename):
        with vistir.contextmanagers.atomic_open_for_write(filename) as fh:
            fh.write("Overwriting all the text from before with even newer text")
            raise RuntimeError("But did it get overwritten now?")

>>> raise_exception_while_writing(fn)
    Traceback (most recent call last):
        ...
    RuntimeError: But did it get overwritten now?

>>> read_test_file()
    writing some new text
```

vistir.contextmanagers.**open_file**(*link*, *session=None*, *stream=True*)

Open local or remote file for reading.

Parameters

- **link** (*pip._internal.index.Link*) – A link object from resolving dependencies with pip, or else a URL.
- **session** (*Optional[Session]*) – A `Session` instance
- **stream** (*bool*) – Whether to stream the content if remote, default True

Raises **ValueError** – If link points to a local directory.

Returns a context manager to the opened file-like object

vistir.contextmanagers.**replaced_stream**(*stream_name*)

Context manager to temporarily swap out *stream_name* with a stream wrapper.

Parameters **stream_name** (*str*) – The name of a sys stream to wrap

Returns A `StreamWrapper` replacement, temporarily

```
>>> orig_stdout = sys.stdout
>>> with replaced_stream("stdout") as stdout:
...     sys.stdout.write("hello")
...     assert stdout.getvalue() == "hello"
```

```
>>> sys.stdout.write("hello")
'hello'
```

vistir.contextmanagers.**replaced_streams**()

Context manager to replace both `sys.stdout` and `sys.stderr` using `replaced_stream`

returns: *(stdout, stderr)*

```
>>> import sys
>>> with vistir.contextmanagers.replaced_streams() as streams:
>>>     stdout, stderr = streams
>>>     sys.stderr.write("test")
>>>     sys.stdout.write("hello")
>>>     assert stdout.getvalue() == "hello"
>>>     assert stderr.getvalue() == "test"
```

```
>>> stdout.getvalue()
'hello'
```

```
>>> stderr.getvalue()
'test'
```

2.1.4 vistir.misc module

vistir.misc.**shell_escape**(*cmd: Union[str, List[str]]*) → str

Escape strings for use in `Popen()` and *run()*.

This is a passthrough method for instantiating a *Script* object which can be used to escape commands to output as a single string.

vistir.misc.**unnest**(*elem*)

Flatten an arbitrarily nested iterable.

Parameters **elem** (`Iterable`) – An iterable to flatten

```
>>> nested_iterable = (
        1234, (3456, 4398345, (234234)), (
            2396, (
                23895750, 9283798, 29384, (
                    289375983275, 293759, 2347, (
                        2098, 7987, 27599
                    )
                )
            )
        )
    )
>>> list(vistir.misc.unnest(nested_iterable))
[1234, 3456, 4398345, 234234, 2396, 23895750, 9283798, 29384, 289375983275,␣
↪293759,
 2347, 2098, 7987, 27599]
```

vistir.misc.**run**(*cmd*, *env=None*, *return_object=False*, *block=True*, *cwd=None*, *verbose=False*, *nospin=False*, *spinner_name=None*, *combine_stderr=True*, *display_limit=200*, *write_to_stdout=True*, *encoding='utf-8'*)

Use *subprocess.Popen* to get the output of a command and decode it.

Parameters

- **cmd** (*list*) – A list representing the command you want to run.
- **env** (*dict*) – Additional environment settings to pass through to the subprocess.
- **return_object** (*bool*) – When True, returns the whole subprocess instance
- **block** (*bool*) – When False, returns a potentially still-running `subprocess.Popen` instance
- **cwd** (*str*) – Current working directory context to use for spawning the subprocess.
- **verbose** (*bool*) – Whether to print stdout in real time when non-blocking.
- **nospin** (*bool*) – Whether to disable the cli spinner.
- **spinner_name** (*str*) – The name of the spinner to use if enabled, defaults to bouncingBar
- **combine_stderr** (*bool*) – Optionally merge stdout and stderr in the subprocess, false if nonblocking.
- **dispay_limit** (*int*) – The max width of output lines to display when using a spinner.
- **write_to_stdout** (*bool*) – Whether to write to stdout when using a spinner, defaults to True.

Returns A 2-tuple of (output, error) or a `subprocess.Popen` object.

> **Warning:** Merging standard out and standard error in a nonblocking subprocess can cause errors in some cases and may not be ideal. Consider disabling this functionality.

vistir.misc.**load_path**(*python*)

Load the `sys.path` from the given python executable's environment as json.

Parameters **python** (*str*) – Path to a valid python executable

Returns A python representation of the *sys.path* value of the given python executable.

Return type list

```
>>> load_path("/home/user/.virtualenvs/requirementslib-5MhGuG3C/bin/python")
['', '/home/user/.virtualenvs/requirementslib-5MhGuG3C/lib/python37.zip',
 '/home/user/.virtualenvs/requirementslib-5MhGuG3C/lib/python3.7',
 '/home/user/.virtualenvs/requirementslib-5MhGuG3C/lib/python3.7/lib-dynload',
 '/home/user/.pyenv/versions/3.7.0/lib/python3.7',
 '/home/user/.virtualenvs/requirementslib-5MhGuG3C/lib/python3.7/site-packages',
 '/home/user/git/requirementslib/src']
```

vistir.misc.**partialclass**(*cls*, **args*, ***kwargs*)

Returns a partially instantiated class.

Returns A partial class instance

Return type cls

```
>>> source = partialclass(Source, url="https://pypi.org/simple")
>>> source
<class '__main__.Source'>
>>> source(name="pypi")
>>> source.__dict__
mappingproxy({
    '__module__': '__main__',
    '__dict__': <attribute '__dict__' of 'Source' objects>,
    '__weakref__': <attribute '__weakref__' of 'Source' objects>,
    '__doc__': None,
    '__init__': functools.partialmethod(
        <function Source.__init__ at 0x7f23af429bf8>, , url='https://pypi.org/
↪simple'
    )
})
>>> new_source = source(name="pypi")
>>> new_source
<__main__.Source object at 0x7f23af189b38>
>>> new_source.__dict__
{'url': 'https://pypi.org/simple', 'verify_ssl': True, 'name': 'pypi'}
```

vistir.misc.**to_text**(*string*, *encoding='utf-8'*, *errors=None*)

Force a value to a text-type.

Parameters

- **string** (*str or bytes unicode*) – Some input that can be converted to a unicode representation.
- **encoding** – The encoding to use for conversions, defaults to "utf-8"
- **encoding** – str, optional

Returns The unicode representation of the string

Return type str

vistir.misc.**to_bytes**(*string*, *encoding='utf-8'*, *errors=None*)

Force a value to bytes.

Parameters

- **string** (*str or bytes unicode or a memoryview subclass*) – Some input that can be converted to a bytes.
- **encoding** – The encoding to use for conversions, defaults to "utf-8"

- **encoding** – str, optional

Returns Corresponding byte representation (for use in filesystem operations)

Return type bytes

vistir.misc.**getpreferredencoding**()
Determine the proper output encoding for terminal rendering.

vistir.misc.**decode_for_output**(*output*, *target_stream=None*, *translation_map=None*)
Given a string, decode it for output to a terminal.

Parameters

- **output** (*str*) – A string to print to a terminal
- **target_stream** – A stream to write to, we will encode to target this stream if possible.
- **translation_map** (*dict*) – A mapping of unicode character ordinals to replacement strings.

Returns A re-encoded string using the preferred encoding

Return type str

vistir.misc.**get_canonical_encoding_name**(*name*)
Given an encoding name, get the canonical name from a codec lookup.

Parameters **name** (*str*) – The name of the codec to lookup

Returns The canonical version of the codec name

Return type str

vistir.misc.**get_wrapped_stream**(*stream*, *encoding=None*, *errors='replace'*)
Given a stream, wrap it in a *StreamWrapper* instance and return the wrapped stream.

Parameters

- **stream** – A stream instance to wrap
- **encoding** (*str*) – The encoding to use for the stream
- **errors** (*str*) – The error handler to use, default "replace"

Returns A new, wrapped stream

Return type *StreamWrapper*

class vistir.misc.**StreamWrapper**(*stream*, *encoding*, *errors*, *line_buffering=True*, ***kwargs*)
Bases: _io.TextIOWrapper

This wrapper class will wrap a provided stream and supply an interface for compatibility.

isatty()
Return whether this is an 'interactive' stream.

Return False if it can't be determined.

write(*x*)
Write string to stream. Returns the number of characters written (which is always equal to the length of the string).

writelines(*lines*)
Write a list of lines to stream.

Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

2.1.5 vistir.path module

vistir.path.**check_for_unc_path**(*path*)
Checks to see if a pathlib *Path* object is a unc path or not.

vistir.path.**get_converted_relative_path**(*path*, *relative_to=None*)
Convert *path* to be relative.

Given a vague relative path, return the path relative to the given location.

Parameters

- **path** (*str*) – The location of a target path
- **relative_to** (*str*) – The starting path to build against, optional

Returns A relative posix-style path with a leading *./*

This performs additional conversion to ensure the result is of POSIX form, and starts with *./*, or is precisely *.*.

```
>>> os.chdir('/home/user/code/myrepo/myfolder')
>>> vistir.path.get_converted_relative_path('/home/user/code/file.zip')
'./../../file.zip'
>>> vistir.path.get_converted_relative_path('/home/user/code/myrepo/myfolder/
↪mysubfolder')
'./mysubfolder'
>>> vistir.path.get_converted_relative_path('/home/user/code/myrepo/myfolder')
'.'
```

vistir.path.**handle_remove_readonly**(*func*, *path*, *exc*)
Error handler for shutil.rmtree.

Windows source repo folders are read-only by default, so this error handler attempts to set them as writeable and then proceed with deletion.

Parameters

- **func** (*function*) – The caller function
- **path** (*str*) – The target path for removal
- **exc** (*Exception*) – The raised exception

This function will call check *is_readonly_path()* before attempting to call *set_write_bit()* on the target path and try again.

vistir.path.**normalize_path**(*path*)
Return a case-normalized absolute variable-expanded path.

Parameters **path** (*str*) – The non-normalized path

Returns A normalized, expanded, case-normalized path

Return type str

vistir.path.**is_in_path**(*path*, *parent*)
Determine if the provided full path is in the given parent root.

Parameters

- **path** (*str*) – The full path to check the location of.
- **parent** (*str*) – The parent path to check for membership in

Returns Whether the full path is a member of the provided parent.

Return type bool

vistir.path.**is_file_url**(*url*)
Returns true if the given url is a file url.

vistir.path.**is_readonly_path**(*fn*)
Check if a provided path exists and is readonly.

Permissions check is *bool(path.stat & stat.S_IREAD)* or *not os.access(path, os.W_OK)*

vistir.path.**is_valid_url**(*url*)
Checks if a given string is an url.

vistir.path.**mkdir_p**(*newdir*, *mode=511*)

vistir.path.**ensure_mkdir_p**(*mode=511*)
Decorator to ensure *mkdir_p* is called to the function's return value.

vistir.path.**create_tracked_tempdir**(**args*, ***kwargs*)
Create a tracked temporary directory.

This uses *TemporaryDirectory*, but does not remove the directory when the return value goes out of scope, instead registers a handler to cleanup on program exit.

The return value is the path to the created directory.

vistir.path.**create_tracked_tempfile**(**args*, ***kwargs*)
Create a tracked temporary file.

This uses the *NamedTemporaryFile* construct, but does not remove the file until the interpreter exits.

The return value is the file object.

vistir.path.**path_to_url**(*path*)
Convert the supplied local path to a file uri.

Parameters **path** (*str*) – A string pointing to or representing a local path

Returns A *file://* uri for the same location

Return type str

```
>>> path_to_url("/home/user/code/myrepo/myfile.zip")
'file:///home/user/code/myrepo/myfile.zip'
```

vistir.path.**rmtree**(*directory: str*, *ignore_errors: bool = False*, *onerror: Optional[Callable] = None*) → None
Stand-in for `rmtree()` with additional error-handling.

This version of *rmtree* handles read-only paths, especially in the case of index files written by certain source control systems.

Parameters

- **directory** (*str*) – The target directory to remove
- **ignore_errors** (*bool*) – Whether to ignore errors, defaults to False
- **onerror** (*func*) – An error handling function, defaults to *handle_remove_readonly()*

Note: Setting *ignore_errors=True* may cause this to silently fail to delete the path

vistir.path.**safe_expandvars**(*value*)
Call os.path.expandvars if value is a string, otherwise do nothing.

vistir.path.**set_write_bit**(*fn: str*) → None
Set read-write permissions for the current user on the target path. Fail silently if the path doesn't exist.

Parameters **fn** (*str*) – The target filename or path

Returns None

vistir.path.**url_to_path**(*url*)
Convert a valid file url to a local filesystem path.

Follows logic taken from pip's equivalent function

vistir.path.**walk_up**(*bottom*)
Mimic os.walk, but walk 'up' instead of down the directory tree.

From: https://gist.github.com/zdavkeos/1098474

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

U

V

W