Reusables Documentation

Release 0.9.3

Chris Griffith

Sep 21, 2018

Contents

Commonly Consumed Code Commodities

CHAPTER 1

Overview

The reusables library is a cookbook of python functions and classes that programmers may find themselves often recreating.

It includes:

- Archiving and extraction for zip, tar, gz, bz2, and rar
- Path (file and folders) management
- Fast logging setup and tools
- Namespace (dict to class modules with child recursion)
- Friendly datetime formatting
- Config to dict parsing
- Common regular expressions and file extensions
- Helpful wrappers
- Bash analogues
- Easy downloading
- Multiprocessing helpers

1.1 Install

Reusables is on PyPI, so can be easily installed with pip or easy_install.

```
pip install reusables
```

There are no required decencies. If this doesn't work, it's broken, raise a github issue.

Reusables is designed to not require any imports outside the standard library, but can be supplemented with those found in the requirements.txt file for additional functionality.

CI tests run on:

- Python 2.6+
- Python 3.3+
- Pypy

Examples are provided below, and the API documentation can always be found at readthedocs.org.

Please note this is currently in development. Any item in development prior to a major version (1.x, 2.x) may change. Once at a major version, no breaking changes are planned to occur within that version.

CHAPTER 2

What's in the box

2.1 General Helpers and File Management

```
import reusables

reusables.find_files_list(".", ext=reusables.exts.pictures)
# ['/home/user/background.jpg', '/home/user/private.png']

reusables.archive("reusables", name="reuse", archive_type="zip")
# 'C:\\Users\\Me\\Reusables\\reuse.zip'

reusables.extract("test/test_structure.zip", "my_archive")
# All files in the zip will be extracted into directory "my_archive"

reusables.config_dict('my_config.cfg')
# {'Section 1': {'key 1': 'value 1', 'key2': 'Value2'}, 'Section 2': {}}

reusables.count_files(".")
# 405

reusables.file_hash("test_structure.zip", hash_type="sha256")
# 'bc11af55928ab89771b9a141fa526a5b8a3dbc7d6870fece9b91af5d345a75ea'

reusables.safe_path('/home/user/eViL User\0\\/newdir$^&*/new^%file.txt')
# '/home/user/eViL User__/newdir____/new__file.txt'

reusables.run("echo 'hello there!'", shell=True)
# CompletedProcess(args="echo 'hello there!'", returncode=0, stdout='hello there!\n')

reusables.cut("abcdefghi")
# ['ab', 'cd', 'ef', 'gh', 'i']
```

One of the most reusables pieces of code is the find_files. It is always appearing on stackoverflow and forums of how to implement os.walk or glob; here's both.

```
reusables.find_files_list(".", name="*reuse*", depth=2)
# ['.\\test\\test_reuse.py', '.\\test\\test_reuse_datetime.py',
#  '.\\test\\test_reuse_logging.py', '.\\test\\test_reuse_namespace.py']

# match_case works for both ext and name
# depth of 1 means this working directory only, no further

reusables.find_files_list(ext=".PY", depth=1, match_case=True)
# []

reusables.find_files_list(ext=".py", depth=1, match_case=True)
# ['.\\setup.py']

reusables.find_files_list(name="setup", ext=".py", match_case=True)
# ['.\\setup.py']

reusables.find_files_list(name="Setup", ext=".py", match_case=True)
# []
```

2.2 Namespace

Check out Box, a much improved version as its own library.

Dictionary management class, similar to Bunch, but designed so that sub-dictionaries are recursively made into namespaces.

```
my_breakfast = {"spam": {"eggs": {"sausage": {"bacon": "yummy"}}}}
namespace_breakfast = reusables.Namespace(**my_breakfast)

print(namespace_breakfast.spam.eggs.sausage.bacon)
# yummy

print(namespace_breakfast.spam.eggs['sausage'].bacon)
# yummy

str(namespace_breakfast['spam'].eggs)
# "{'sausage': {'bacon': 'yummy'}}"

namespace_breakfast.to_dict()
#{'spam': {'eggs': {'sausage': {'bacon': 'yummy'}}}}

dict(namespace_breakfast)
# {'spam': <Namespace: {'eggs': {'sausage': {'bacon': '...>}
# This is NOT the same as .to_dict() as it is not recursive
```

2.3 Logging

```
logger = reusables.setup_logger(__name__)
# By default it adds a stream logger to sys.stderr

logger.info("Test")
# 2016-04-25 19:32:45,542 __main__     INFO     Test
```

There are multiple log formatters provided, as well as additional helper functions. All helper functions will accept either the logger object or the name of the logger.

```
reusables.remove_stream_handlers(__name__)
# remove_file_handlers() and remove_all_handlers() also available

reusables.add_stream_handler(__name__, log_format=reusables.log_formats.detailed)
r.add_rotating_file_handler(__name__, "my.log", level=logging.INFO)

logger.info("Example log entry")
# 2016-12-14 20:56:55,446 : 315147 MainThread : reusables.log INFO Example log entry

open("my.log").read()
# 2016-12-14 20:56:55,446 - __builtin__    INFO     Example log entry
```

Provided log formats

Feel free to provide your own formats, aided by the docs. However this includes some commonly used ones you may find useful. they are all stored in the Namespace "reusables.log_formats" (feel free to use it as a dict as stated above).

Because ReStructuredText tables don't preserve whitespace (even with literals), which is important to show distinction in these formatters, here's it in a code block instead.

```
reusables.log_formats.keys()
# ['common', 'level_first', 'threaded', 'easy_read', 'easy_thread', 'detailed']

logger = reusables.setup_logger(__name__, log_format=reusables.log_formats.threaded)
reusables.add_timed_rotating_file_handler(logger, "timed.log", level=logging.ERROR,␣
↪log_format=reusables.log_formats.detailed)
```

```
+--------------+-------------------------------------------------------------------------
↪----------------+
| Formatter    | Example Output                                                          ␣
↪                |
+==============+=========================================================================
| easy_read    | 2016-04-26 21:17:51,225 - example_logger  INFO      example log␣
↪message              |
|              | 2016-04-26 21:17:59,074 - example_logger  ERROR     Something broke ␣
↪                |
+--------------+-------------------------------------------------------------------------
↪----------------+
| detailed     | 2016-04-26 21:17:51,225 :  7020 MainThread : example_logger INFO␣
↪example log message |
|              | 2016-04-26 21:17:59,074 : 14868 MainThread : example_logger ERROR␣
↪Something broke    |
+--------------+-------------------------------------------------------------------------
↪----------------+
| level_first  | INFO - example_logger - 2016-04-26 21:17:51,225 - example log␣
↪message                |
|              | ERROR - example_logger - 2016-04-26 21:17:59,074 - Something broke  ␣
↪                |
+--------------+-------------------------------------------------------------------------
↪----------------+
| threaded     | 7020 MainThread : example log message                                   ␣
↪                |
|              | 14868 MainThread : Something broke                                      ␣
↪                |
+--------------+-------------------------------------------------------------------------
↪----------------+
```

(continues on next page)

(continued from previous page)

```
| easy_thread   |  7020 MainThread : example_logger  INFO      example log message    |
|               | 14868 MainThread : example_logger  ERROR     Something broke        |
+---------------+-----------------------------------------------------------------+
| common        | 2016-04-26 21:17:51,225 - example_logger - INFO - example log message |
|               | 2016-04-26 21:17:59,074 - example_logger - ERROR - Something broke    |
+---------------+-----------------------------------------------------------------+
```

2.4 Extension Groups

It's common to be looking for a specific type of file.

```
if file_path.endswith(reusables.exts.pictures):
    print("{} is a picture file".format(file_path))
```

That's right, str.endswith (as well as str.startswith) accept a tuple to search.

File Type	Extensions
pictures	.jpeg .jpg .png .gif .bmp .tif .tiff .ico .mng .tga .psd .xcf .svg .icns
video	.mkv .avi .mp4 .mov .flv .mpeg .mpg .3gp .m4v .ogv .asf .m1v .m2v .mpe .ogv .wmv .rm .qt .3g2 .asf .vob
music	.mp3 .ogg .wav .flac .aif .aiff .au .m4a .wma .mp2 .m4a .m4p .aac .ra .mid .midi .mus .psf
documents	.doc .docx .pdf .xls .xlsx .ppt .pptx .csv .epub .gdoc .odt .rtf .txt .info .xps .gslides .gsheet .pages .msg .tex .wpd .wps .csv
archives	.zip .rar .7z .tar.gz .tgz .gz .bzip .bzip2 .bz2 .xz .lzma .bin .tar
cd_images	.iso .nrg .img .mds .mdf .cue .daa
scripts	.py .sh .bat
binaries	.msi .exe
markup	.html .htm .xml .yaml .json .raml .xhtml .kml

2.5 Wrappers

unique

There are tons of wrappers for caching and saving inputs and outputs, this is a different take that requires the function returns a result not yet provided.

```
@reusables.unique(max_retries=100, error_text="All UIDs taken!")
def gen_small_uid():
    return random.randint(0, 100)
```

time_it

Easily time the execution time of a function, using the high precision perf_conuter on Python 3.3+, otherwise clock.

```
@reusables.time_it()
def test_it():
    return time.sleep(float(f"0.{random.randint(1, 9)}"))
```

CHAPTER 3

Command line helpers

Use the Python interpreter as much as a shell? Here's some handy helpers to fill the void. (Please don't do 'import *' in production code, this is used as an easy to use example using the interpreter interactively.)

> These are not imported by default with "import reusables", as they are designed to be imported only in an interactive shell

Some commands from other areas are also included where they are highly applicable in both instances, such as 'touch' and 'download'.

```
from reusables.cli import *

cd("~") # Automatic user expansion unlike os.chdir()

pwd()
# '/home/user'

pushd("Downloads")
# ['Downloads', '/home/user']

pwd()
# '/home/user/Downloads'

popd()
# ['/home/user']

ls("-lah")  # Uses 'ls' on linux and 'dir' on windows
#  total 1.5M
#  drwxr-xr-x 49 james james 4.0K Nov  1 20:09 .
#  drwxr-xr-x  3 root  root  4.0K Aug 21  2015 ..
#  -rw-rw-r--  1 james james  22K Aug 22 13:21 picture.jpg
#  -rw-------  1 james james  17K Nov  1 20:08 .bash_history

cmd("ifconfig") # Shells, decodes and prints 'reusables.run' output
#   eth0      Link encap:Ethernet  HWaddr de:ad:be:ef:00:00
#             inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
```

(continues on next page)

(continued from previous page)

```
#             ...

download('https://www.python.org/ftp/python/README.html', save_to_file=False)
# 2016-11-02 10:37:23,644 - reusables.web  INFO      Downloading https://www.python.
↪org/ftp/python/README.html (2.3 KB) to memory
# b'<PRE>\nPython Distribution...
```

3.1 DateTime

Easy formatting for datetime objects. Also parsing for ISO formatted time.

```
reusables.datetime_format("Wake up {son}, it's {hours}:{minutes} {periods}!"
                          "I don't care if it's a {day-fullname}, {command}!",
                          son="John",
                          command="Get out of bed!")
# "Wake up John, it's 09:51 AM! I don't care if it's a Saturday, Get out of bed!!"

reusables.datetime_from_iso('2017-03-10T12:56:55.031863')
# datetime.datetime(2017, 3, 10, 12, 56, 55, 31863)
```

Examples based on Mon Mar 28 13:27:11 2016

Format	Mapping	Example
{12-hour}	%I	01
{24-hour}	%H	13
{seconds}	%S	14
{minutes}	%M	20
{microseconds}	%f	320944
{time-zone}	%Z	
{years}	%y	16
{years-full}	%Y	2016
{months}	%m	03
{months-name}	%b	Mar
{months-full}	%B	March
{days}	%d	28
{week-days}	%w	1
{year-days}	%j	088
{days-name}	%a	Mon
{days-full}	%A	Monday
{mon-weeks}	%W	13
{date}	%x	03/28/16
{time}	%X	13:27:11
{date-time}	%C	Mon Mar 28 13:27:11 2016
{utc-offset}	%Z	
{periods}	%p	PM
{iso-format}	%Y-%m-%dT%H:%M:%S	2016-03-28T13:27:11

CHAPTER 4

FAQ

How can I help? / Why doesn't it do what I want it too?

Please feel free to make suggestions in the 'issues' section of github, or to be super duper helpful go ahead and submit a PR for the functionality you want to see! Only requirements are that it's well thought out and is more in place here rather than it's own project (to be merged will need documentation and basic unittests as well, but not a requirement for opening the PR). Please don't hesitate if you're new to python! Even the smallest PR contributions will earn a mention in a brand new Contributors section.

Unrar not installed?

A common error to see, especially on Windows based systems, is: "rarfile.RarCannotExec: Unrar not installed? (rarfile.UNRAR_TOOL='unrar')"

This is probably because unrar is not downloaded or linked properly. Download UnRAR from http://www.rarlab.com/rar_add.htm and follow these instructions before trying again: http://rarfile.readthedocs.org/en/latest/faq.html?highlight=windows#how-can-i-get-it-work-on-windows

CHAPTER 5

License

CHAPTER 6

Additional Info

This does not claim to provide the most accurate, fastest or most 'pythonic' way to implement these useful snippets, this is simply designed for easy reference. Any contributions that would help add functionality or improve existing code is warmly welcomed!

6.1 File Operations

load_json(*json_file*, ***kwargs*)

Open and load data from a JSON file

```
reusables.load_json("example.json")
# {u'key_1': u'val_1', u'key_for_dict': {u'sub_dict_key': 8}}
```

Parameters

- **json_file** – Path to JSON file as string
- **kwargs** – Additional arguments for the json.load command

Returns Dictionary

list_to_csv(*my_list*, *csv_file*)

Save a matrix (list of lists) to a file as a CSV

```
my_list = [["Name", "Location"],
           ["Chris", "South Pole"],
           ["Harry", "Depth of Winter"],
           ["Bob", "Skull"]]

reusables.list_to_csv(my_list, "example.csv")
```

example.csv

Parameters

- **my_list** – list of lists to save to CSV
- **csv_file** – File to save data to

save_json(*data*, *json_file*, *indent=4*, ***kwargs*)
Takes a dictionary and saves it to a file as JSON

```
my_dict = {"key_1": "val_1",
           "key_for_dict": {"sub_dict_key": 8}}

reusables.save_json(my_dict,"example.json")
```

example.json

```
{
    "key_1": "val_1",
    "key_for_dict": {
        "sub_dict_key": 8
    }
}
```

Parameters

- **data** – dictionary to save as JSON
- **json_file** – Path to save file location as str
- **indent** – Format the JSON file with so many numbers of spaces
- **kwargs** – Additional arguments for the json.dump command

csv_to_list(*csv_file*)
Open and transform a CSV file into a matrix (list of lists).

```
reusables.csv_to_list("example.csv")
# [['Name', 'Location'],
#  ['Chris', 'South Pole'],
#  ['Harry', 'Depth of Winter'],
#  ['Bob', 'Skull']]
```

Parameters **csv_file** – Path to CSV file as str

Returns list

extract(*archive_file*, *path='.'*, *delete_on_success=False*, *enable_rar=False*)
Automatically detect archive type and extract all files to specified path.

```
import os

os.listdir(".")
# ['test_structure.zip']

reusables.extract("test_structure.zip")

os.listdir(".")
# [ 'test_structure', 'test_structure.zip']
```

Parameters

- **archive_file** – path to file to extract
- **path** – location to extract to
- **delete_on_success** – Will delete the original archive if set to True
- **enable_rar** – include the rarfile import and extract

Returns path to extracted files

archive(*files_to_archive*, *name='archive.zip'*, *archive_type=None*, *overwrite=False*, *store=False*, *depth=None*, *err_non_exist=True*, *allow_zip_64=True*, ***tarfile_kwargs*)

Archive a list of files (or files inside a folder), can chose between

- zip
- tar
- gz (tar.gz, tgz)
- bz2 (tar.bz2)

```
reusables.archive(['reusables', '.travis.yml'],
                  name="my_archive.bz2")
# 'C:\Users\Me\Reusables\my_archive.bz2'
```

Parameters

- **files_to_archive** – list of files and folders to archive
- **name** – path and name of archive file
- **archive_type** – auto-detects unless specified
- **overwrite** – overwrite if archive exists
- **store** – zipfile only, True will not compress files
- **depth** – specify max depth for folders
- **err_non_exist** – raise error if provided file does not exist
- **allow_zip_64** – must be enabled for zip files larger than 2GB
- **tarfile_kwargs** – extra args to pass to tarfile.open

Returns path to created archive

config_dict(*config_file=None*, *auto_find=False*, *verify=True*, ***cfg_options*)

Return configuration options as dictionary. Accepts either a single config file or a list of files. Auto find will search for all .cfg, .config and .ini in the execution directory and package root (unsafe but handy).

```
reusables.config_dict(os.path.join("test", "data", "test_config.ini"))
# {'General': {'example': 'A regular string'},
#  'Section 2': {'anint': '234',
#                'examplelist': '234,123,234,543',
#                'floatly': '4.4',
#                'my_bool': 'yes'}}
```

Parameters

- **config_file** – path or paths to the files location
- **auto_find** – look for a config type file at this location or below

- **verify** – make sure the file exists before trying to read
- **cfg_options** – options to pass to the parser

Returns dictionary of the config files

config_namespace(*config_file=None*, *auto_find=False*, *verify=True*, ***cfg_options*)

Return configuration options as a Namespace.

```
reusables.config_namespace(os.path.join("test", "data",
                                        "test_config.ini"))
# <Namespace: {'General': {'example': 'A regul...>
```

Parameters

- **config_file** – path or paths to the files location
- **auto_find** – look for a config type file at this location or below
- **verify** – make sure the file exists before trying to read
- **cfg_options** – options to pass to the parser

Returns Namespace of the config files

os_tree(*directory*, *enable_scandir=False*)

Return a directories contents as a dictionary hierarchy.

```
reusables.os_tree(".")
# {'doc': {'build': {'doctrees': {},
#                    'html': {'_sources': {}, '_static': {}}},
#          'source': {}},
#  'reusables': {'__pycache__': {}},
#  'test': {'__pycache__': {}, 'data': {}}}
```

Parameters

- **directory** – path to directory to created the tree of.
- **enable_scandir** – on python < 3.5 enable external scandir package

Returns dictionary of the directory

check_filename(*filename*)

Returns a boolean stating if the filename is safe to use or not. Note that this does not test for "legal" names accepted, but a more restricted set of: Letters, numbers, spaces, hyphens, underscores and periods.

Parameters **filename** – name of a file as a string

Returns boolean if it is a safe file name

count_files(**args*, ***kwargs*)

Returns an integer of all files found using find_files

directory_duplicates(*directory*, *hash_type='md5'*, ***kwargs*)

Find all duplicates in a directory. Will return a list, in that list are lists of duplicate files.

Parameters

- **directory** – Directory to search
- **hash_type** – Type of hash to perform

- **kwargs** – Arguments to pass to find_files to narrow file types

Returns list of lists of dups

dup_finder(*file_path*, *directory='.'*, *enable_scandir=False*)

Check a directory for duplicates of the specified file. This is meant for a single file only, for checking a directory for dups, use directory_duplicates.

This is designed to be as fast as possible by doing lighter checks before progressing to more extensive ones, in order they are:

1. File size
2. First twenty bytes
3. Full SHA256 compare

```
list(reusables.dup_finder(
     "test_structure\files_2\empty_file"))
# ['C:\Reusables\test\data\fake_dir',
#  'C:\Reusables\test\data\test_structure\Files\empty_file_1',
#  'C:\Reusables\test\data\test_structure\Files\empty_file_2',
#  'C:\Reusables\test\data\test_structure\files_2\empty_file']
```

Parameters

- **file_path** – Path to file to check for duplicates of
- **directory** – Directory to dig recursively into to look for duplicates
- **enable_scandir** – on python < 3.5 enable external scandir package

Returns generators

file_hash(*path*, *hash_type='md5'*, *block_size=65536*, *hex_digest=True*)

Hash a given file with md5, or any other and return the hex digest. You can run *hashlib.algorithms_available* to see which are available on your system unless you have an archaic python version, you poor soul).

This function is designed to be non memory intensive.

```
reusables.file_hash(test_structure.zip")
# '61e387de305201a2c915a4f4277d6663'
```

Parameters

- **path** – location of the file to hash
- **hash_type** – string name of the hash to use
- **block_size** – amount of bytes to add to hasher at a time
- **hex_digest** – returned as hexdigest, false will return digest

Returns file's hash

find_files(*directory='.'*, *ext=None*, *name=None*, *match_case=False*, *disable_glob=False*, *depth=None*, *abspath=False*, *enable_scandir=False*)

Walk through a file directory and return an iterator of files that match requirements. Will autodetect if name has glob as magic characters.

Note: For the example below, you can use find_files_list to return as a list, this is simply an easy way to show the output.

```
list(reusables.find_files(name="ex", match_case=True))
# ['C:\example.pdf',
#  'C:\My_exam_score.txt']

list(reusables.find_files(name="*free*"))
# ['C:\my_stuff\Freedom_fight.pdf']

list(reusables.find_files(ext=".pdf"))
# ['C:\Example.pdf',
#  'C:\how_to_program.pdf',
#  'C:\Hunks_and_Chicks.pdf']

list(reusables.find_files(name="*chris*"))
# ['C:\Christmas_card.docx',
#  'C:\chris_stuff.zip']
```

Parameters

- **directory** – Top location to recursively search for matching files
- **ext** – Extensions of the file you are looking for
- **name** – Part of the file name
- **match_case** – If name or ext has to be a direct match or not
- **disable_glob** – Do not look for globable names or use glob magic check
- **depth** – How many directories down to search
- **abspath** – Return files with their absolute paths
- **enable_scandir** – on python < 3.5 enable external scandir package

Returns generator of all files in the specified directory

find_files_list(**args*, ***kwargs*)

Returns a list of find_files generator

join_here(**paths*, ***kwargs*)

Join any path or paths as a sub directory of the current file's directory.

```
reusables.join_here("Makefile")
# 'C:\Reusables\Makefile'
```

Parameters

- **paths** – paths to join together
- **kwargs** – 'strict', do not strip os.sep
- **kwargs** – 'safe', make them into a safe path it True

Returns abspath as string

join_paths(**paths*, ***kwargs*)

Join multiple paths together and return the absolute path of them. If 'safe' is specified, this function will 'clean' the path with the 'safe_path' function. This will clean root decelerations from the path after the first item.

Would like to do 'safe=False' instead of '**kwargs' but stupider versions of python *cough 2.6* don't like that after '*paths'.

Parameters

- **paths** – paths to join together
- **kwargs** – 'safe', make them into a safe path it True

Returns abspath as string

remove_empty_directories(*root_directory*, *dry_run=False*, *ignore_errors=True*, *enable_scandir=False*)

Remove all empty folders from a path. Returns list of empty directories.

Parameters

- **root_directory** – base directory to start at
- **dry_run** – just return a list of what would be removed
- **ignore_errors** – Permissions are a pain, just ignore if you blocked
- **enable_scandir** – on python < 3.5 enable external scandir package

Returns list of removed directories

remove_empty_files(*root_directory*, *dry_run=False*, *ignore_errors=True*, *enable_scandir=False*)

Remove all empty files from a path. Returns list of the empty files removed.

Parameters

- **root_directory** – base directory to start at
- **dry_run** – just return a list of what would be removed
- **ignore_errors** – Permissions are a pain, just ignore if you blocked
- **enable_scandir** – on python < 3.5 enable external scandir package

Returns list of removed files

safe_filename(*filename*, *replacement='_'*)

Replace unsafe filename characters with underscores. Note that this does not test for "legal" names accepted, but a more restricted set of: Letters, numbers, spaces, hyphens, underscores and periods.

Parameters

- **filename** – name of a file as a string
- **replacement** – character to use as a replacement of bad characters

Returns safe filename string

safe_path(*path*, *replacement='_'*)

Replace unsafe path characters with underscores. Do NOT use this with existing paths that cannot be modified, this to to help generate new, clean paths.

Supports windows and *nix systems.

Parameters

- **path** – path as a string
- **replacement** – character to use in place of bad characters

Returns a safer path

touch(*path*)

Native 'touch' functionality in python

Parameters **path** – path to file to 'touch'

6.2 Tasker

class Tasker(*tasks=()*, *max_tasks=4*, *task_timeout=None*, *task_queue=None*, *result_queue=None*, *command_queue=None*, *run_until=None*, *logger='reusables'*, ***task_kwargs*)

An advanced multiprocessing pool, designed to run in the background, with ability to change number of workers or pause the service all together.

Simply subclass Tasker, overwrite *perform_task* and *run*!

It has in and out queues (task_queue, result_queue) which can be provided or will automatically be created as 'multiprocessing.Queue()'s.

> **Warning:** Use multiprocessing.Queue not queue.Queue

> **Warning:** Do not use on Windows at this time

Parameters

- **tasks** – list of tasks to pre-populate the queue with
- **max_tasks** – the max number of parallel workers
- **task_timeout** – how long can each task take
- **task_queue** – option to specify an existing queue of tasks
- **result_queue** – option to specify an existing queue for results
- **run_until** – datetime to run until

change_task_size(*size*)
Blocking request to change number of running tasks

get(*timeout=None*)
Retrieve next result from the queue

get_state()
Get general information about the state of the class

hook_post_command()

hook_post_task()

hook_pre_command()

hook_pre_task()

main_loop(*stop_at_empty=False*)
Blocking function that can be run directly, if so would probably want to specify 'stop_at_empty' to true, or have a separate process adding items to the queue.

pause()
Stop any more tasks from being run

static perform_task(*task*, *result_queue*, ***kwargs*)
Function to be overwritten that performs the tasks from the list

put(*task*)
Add a task to be processed to the queue

run()
Start the main loop as a background process. *nix only

stop()
Hard stop the server and sub process

unpuase()
Allows tasks to be run again

6.3 Logging

Logging helper functions and common log formats.

get_logger(**args*, ***kwargs*)
Depreciated, use setup_logger

get_registered_loggers(*hide_children=False*, *hide_reusables=False*)
Find the names of all loggers currently registered

Parameters

- **hide_children** – only return top level logger names
- **hide_reusables** – hide the reusables loggers

Returns list of logger names

get_file_handler(*file_path='out.log'*, *level=20*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*, *handler=<class 'logging.FileHandler'>*, ***handler_kwargs*)
Set up a file handler to add to a logger.

Parameters

- **file_path** – file to write the log to, defaults to out.log
- **level** – logging level to set handler at
- **log_format** – formatter to use
- **handler** – logging handler to use, defaults to FileHandler
- **handler_kwargs** – options to pass to the handler

Returns handler

get_stream_handler(*stream=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *level=20*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*)
Returns a set up stream handler to add to a logger.

Parameters

- **stream** – which stream to use, defaults to sys.stderr
- **level** – logging level to set handler at
- **log_format** – formatter to use

Returns stream handler

add_file_handler(*logger=None*, *file_path='out.log'*, *level=20*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*)
Addes a newly created file handler to the specified logger

Parameters

- **logger** – logging name or object to modify, defaults to root logger
- **file_path** – path to file to log to
- **level** – logging level to set handler at
- **log_format** – formatter to use

add_stream_handler(*logger=None*, *stream=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *level=20*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*)

Addes a newly created stream handler to the specified logger

Parameters

- **logger** – logging name or object to modify, defaults to root logger
- **stream** – which stream to use, defaults to sys.stderr
- **level** – logging level to set handler at
- **log_format** – formatter to use

add_rotating_file_handler(*logger=None*, *file_path='out.log'*, *level=20*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*, *max_bytes=10485760*, *backup_count=5*, ***handler_kwargs*)

Adds a rotating file handler to the specified logger.

Parameters

- **logger** – logging name or object to modify, defaults to root logger
- **file_path** – path to file to log to
- **level** – logging level to set handler at
- **log_format** – log formatter
- **max_bytes** – Max file size in bytes before rotating
- **backup_count** – Number of backup files
- **handler_kwargs** – options to pass to the handler

add_timed_rotating_file_handler(*logger=None*, *file_path='out.log'*, *level=20*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*, *when='w0'*, *interval=1*, *backup_count=5*, ***handler_kwargs*)

Adds a timed rotating file handler to the specified logger. Defaults to weekly rotation, with 5 backups.

Parameters

- **logger** – logging name or object to modify, defaults to root logger
- **file_path** – path to file to log to
- **level** – logging level to set handler at
- **log_format** – log formatter
- **when** –
- **interval** –
- **backup_count** – Number of backup files
- **handler_kwargs** – options to pass to the handler

change_logger_levels(*logger=None*, *level=10*)
Go through the logger and handlers and update their levels to the one specified.

Parameters

- **logger** – logging name or object to modify, defaults to root logger
- **level** – logging level to set at (10=Debug, 20=Info, 30=Warn, 40=Error)

remove_all_handlers(*logger=None*)
Safely remove all handlers from the logger

Parameters **logger** – logging name or object to modify, defaults to root logger

remove_file_handlers(*logger=None*)
Remove only file handlers from the specified logger. Will go through and close each handler for safety.

Parameters **logger** – logging name or object to modify, defaults to root logger

remove_stream_handlers(*logger=None*)
Remove only stream handlers from the specified logger

Parameters **logger** – logging name or object to modify, defaults to root logger

setup_logger(*module_name=None*, *level=20*, *stream=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *file_path=None*, *log_format='%(asctime)s - %(name)-12s %(levelname)-8s %(message)s'*, *suppress_warning=True*)
Grabs the specified logger and adds wanted handlers to it. Will default to adding a stream handler.

Parameters

- **module_name** – logger name to use
- **level** – logging level to set logger at
- **stream** – stream to log to, or None
- **file_path** – file path to log to, or None
- **log_format** – format to set the handlers to use
- **suppress_warning** – add a NullHandler if no other handler is specified

Returns configured logger

6.4 DateTime

now(*utc=False*, *tz=None*)
Get a current DateTime object. By default is local.

```
reusables.now()
# DateTime(2016, 12, 8, 22, 5, 2, 517000)

reusables.now().format("It's {24-hour}:{min}")
# "It's 22:05"
```

Parameters

- **utc** – bool, default False, UTC time not local
- **tz** – TimeZone as specified by the datetime module

Returns reusables.DateTime

datetime_format(*desired_format*, *datetime_instance=None*, **args*, ***kwargs*)

Replaces format style phrases (listed in the dt_exps dictionary) with this datetime instance's information.

```
reusables.datetime_format("Hey, it's {month-full} already!")
"Hey, it's March already!"
```

Parameters

- **desired_format** – string to add datetime details too
- **datetime_instance** – datetime.datetime instance, defaults to 'now'
- **args** – additional args to pass to str.format
- **kwargs** – additional kwargs to pass to str format

Returns formatted string

datetime_from_iso(*iso_string*)

Create a DateTime object from a ISO string

```
reusables.datetime_from_iso('2017-03-10T12:56:55.031863')
datetime.datetime(2017, 3, 10, 12, 56, 55, 31863)
```

Parameters **iso_string** – string of an ISO datetime

Returns DateTime object

dtf(*desired_format*, *datetime_instance=None*, **args*, ***kwargs*)

Replaces format style phrases (listed in the dt_exps dictionary) with this datetime instance's information.

```
reusables.datetime_format("Hey, it's {month-full} already!")
"Hey, it's March already!"
```

Parameters

- **desired_format** – string to add datetime details too
- **datetime_instance** – datetime.datetime instance, defaults to 'now'
- **args** – additional args to pass to str.format
- **kwargs** – additional kwargs to pass to str format

Returns formatted string

dtiso(*iso_string*)

Create a DateTime object from a ISO string

```
reusables.datetime_from_iso('2017-03-10T12:56:55.031863')
datetime.datetime(2017, 3, 10, 12, 56, 55, 31863)
```

Parameters **iso_string** – string of an ISO datetime

Returns DateTime object

6.5 Namespace

Improved dictionary management. Inspired by javascript style referencing, as it's one of the few things they got right.

class Namespace(**args*, ***kwargs*)

Namespace container. Allows access to attributes by either class dot notation or item reference

All valid:

- namespace.spam.eggs
- namespace['spam']['eggs']
- namespace['spam'].eggs

to_dict(*in_dict=None*)

Turn the Namespace and sub Namespaces back into a native python dictionary.

Parameters **in_dict** – Do not use, for self recursion

Returns python dictionary of this Namespace

tree_view(*sep=' '*)

class ConfigNamespace(**args*, ***kwargs*)

Modified namespace object to add object transforms.

Allows for build in transforms like:

cns = ConfigNamespace(my_bool='yes', my_int='5', my_list='5,4,3,3,2')

cns.bool('my_bool') # True cns.int('my_int') # 5 cns.list('my_list', mod=lambda x: int(x)) # [5, 4, 3, 3, 2]

bool(*item*, *default=None*)

Return value of key as a boolean

Parameters

- **item** – key of value to transform
- **default** – value to return if item does not exist

Returns approximated bool of value

float(*item*, *default=None*)

Return value of key as a float

Parameters

- **item** – key of value to transform
- **default** – value to return if item does not exist

Returns float of value

getboolean(*item*, *default=None*)

getfloat(*item*, *default=None*)

getint(*item*, *default=None*)

int(*item*, *default=None*)

Return value of key as an int

Parameters

- **item** – key of value to transform

- **default** – value to return if item does not exist

Returns int of value

list(*item*, *default=None*, *spliter=', '*, *strip=True*, *mod=None*)
Return value of key as a list

Parameters

- **item** – key of value to transform
- **mod** – function to map against list
- **default** – value to return if item does not exist
- **spliter** – character to split str on
- **strip** – clean the list with the *strip*

Returns list of items

class ProtectedDict

A special dict class that prohibits the setting of keys and attributes. It will NOT protect objects stored in the dictionary, such as sub dicts.

amespace.py", line 249, in __setitem__ # AttributeError: This is a protected dict, cannot change anything

ns
alias of *reusables.namespace.Namespace*

cns
alias of *reusables.namespace.ConfigNamespace*

6.6 Web Based Helpers

download(*url*, *save_to_file=True*, *save_dir='.'*, *filename=None*, *block_size=64000*, *overwrite=False*, *quiet=False*)
Download a given URL to either file or memory

Parameters

- **url** – Full url (with protocol) of path to download
- **save_to_file** – boolean if it should be saved to file or not
- **save_dir** – location of saved file, default is current working dir
- **filename** – filename to save as
- **block_size** – download chunk size
- **overwrite** – overwrite file if it already exists
- **quiet** – boolean to turn off logging for function

Returns save location (or content if not saved to file)

class ThreadedServer(*name=''*, *port=8080*, *auto_start=True*, *server=<class 'http.server.HTTPServer'>*, *handler=<class 'http.server.SimpleHTTPRequestHandler'>*)
Defaulting as a FileServer, this class allows for fast creation of a threaded server that is easily stoppable.

```
my_server = reusables.ThreadedServer()
reusables.download("http://localhost:8080", False)
# '...<html><title>Directory listing for /</title>...'
my_server.stop()
```

Parameters

- **name** – server name
- **port** – int of port to run server on (below 1024 requires root)
- **auto_start** – automatically start the background thread and serve
- **server** – Default is TCPServer (py2) or HTTPServer (py3)
- **handler** – Default is SimpleHTTPRequestHandler

start()
Create a background thread for httpd and serve 'forever'

stop()
Stop the httpd server and join the thread

url_to_ip(*url*)
Provide IP of host, does not support IPv6, uses *socket.gethostbyaddr*

```
reusables.url_to_ip('example.com')
# '93.184.216.34'
```

Parameters **url** – hostname to resolve to IP addresses

Returns string of IP address or None

url_to_ips(*url*, *port=None*, *ipv6=False*, *connect_type=<SocketKind.SOCK_STREAM: 1>*, *proto=6*, *flags=0*)
Provide a list of IP addresses, uses *socket.getaddrinfo*

```
reusables.url_to_ips("example.com", ipv6=True)
# ['2606:2800:220:1:248:1893:25c8:1946']
```

Parameters

- **url** – hostname to resolve to IP addresses
- **port** – port to send to getaddrinfo
- **ipv6** – Return IPv6 address if True, otherwise IPv4
- **connect_type** – defaults to STREAM connection, can be 0 for all
- **proto** – defaults to TCP, can be 0 for all
- **flags** – additional flags to pass

Returns list of resolved IPs

ip_to_url(*ip_addr*)
Resolve a hostname based off an IP address.

This is very limited and will probably not return any results if it is a shared IP address or an address with improperly setup DNS records.

```
reusables.ip_to_url('93.184.216.34') # example.com
# None

reusables.ip_to_url('8.8.8.8')
# 'google-public-dns-a.google.com'
```

Parameters **ip_addr** – IP address to resolve to hostname

Returns string of hostname or None

6.7 Multiprocess Helpers

run(*command*, *input=None*, *stdout=-1*, *stderr=-1*, *timeout=None*, *copy_local_env=False*, ***kwargs*)
Cross platform compatible subprocess with CompletedProcess return.

No formatting or encoding is performed on the output of subprocess, so it's output will appear the same on each version / interpreter as before.

```
reusables.run('echo "hello world!', shell=True)
# CPython 3.6
# CompletedProcess(args='echo "hello world!', returncode=0,
#                   stdout=b'"hello world!\r\n', stderr=b'')
#
# PyPy 5.4 (Python 2.7.10)
# CompletedProcess(args='echo "hello world!', returncode=0L,
# stdout='"hello world!\r\n')
```

Timeout is only usable in Python 3.X, as it was not implemented before then, a NotImplementedError will be raised if specified on 2.x version of Python.

Parameters

- **command** – command to run, str if shell=True otherwise must be list
- **input** – send something *communicate*
- **stdout** – PIPE or None
- **stderr** – PIPE or None
- **timeout** – max time to wait for command to complete
- **copy_local_env** – Use all current ENV vars in the subprocess as well
- **kwargs** – additional arguments to pass to Popen

Returns CompletedProcess class

run_in_pool(*target*, *iterable*, *threaded=True*, *processes=4*, *asynchronous=False*, *target_kwargs=None*)
Run a set of iterables to a function in a Threaded or MP Pool.

Parameters

- **target** – function to run
- **iterable** – positional arg to pass to function
- **threaded** – Threaded if True multiprocessed if False
- **processes** – Number of workers

- **asynchronous** – will do map_async if True
- **target_kwargs** – Keyword arguments to set on the function as a partial

Returns pool results

6.8 Command Line Helpers

Functions to only be in an interactive instances to ease life.

cmd(*command*, *ignore_stderr=False*, *raise_on_return=False*, *timeout=None*, *encoding='utf-8'*)
Run a shell command and have it automatically decoded and printed

Parameters

- **command** – Command to run as str
- **ignore_stderr** – To not print stderr
- **raise_on_return** – Run CompletedProcess.check_returncode()
- **timeout** – timeout to pass to communicate if python 3
- **encoding** – How the output should be decoded

pushd(*directory*)
Change working directories in style and stay organized!

Parameters **directory** – Where do you want to go and remember?

Returns saved directory stack

popd()
Go back to where you once were.

Returns saved directory stack

pwd()
Get the current working directory

cd(*directory*)
Change working directory, with built in user (~) expansion

Parameters **directory** – New place you wanted to go

ls(*params=''*, *directory='.'*, *printed=True*)
Know the best python implantation of ls? It's just to subprocess ls... (uses dir on windows).

Parameters

- **params** – options to pass to ls or dir
- **directory** – if not this directory
- **printed** – If you're using this, you probably wanted it just printed

Returns if not printed, you can parse it yourself

find(*name=None*, *ext=None*, *directory='.'*, *match_case=False*, *disable_glob=False*, *depth=None*)
Designed for the interactive interpreter by making default order of find_files faster.

Parameters

- **name** – Part of the file name
- **ext** – Extensions of the file you are looking for

- **directory** – Top location to recursively search for matching files
- **match_case** – If name has to be a direct match or not
- **disable_glob** – Do not look for globable names or use glob magic check
- **depth** – How many directories down to search

Returns list of all files in the specified directory

head(*file_path*, *lines=10*, *encoding='utf-8'*, *printed=True*, *errors='strict'*)
Read the first N lines of a file, defaults to 10

Parameters

- **file_path** – Path to file to read
- **lines** – Number of lines to read in
- **encoding** – defaults to utf-8 to decode as, will fail on binary
- **printed** – Automatically print the lines instead of returning it
- **errors** – Decoding errors: 'strict', 'ignore' or 'replace'

Returns if printed is false, the lines are returned as a list

cat(*file_path*, *encoding='utf-8'*, *errors='strict'*)

Parameters

- **file_path** – Path to file to read
- **encoding** – defaults to utf-8 to decode as, will fail on binary
- **errors** – Decoding errors: 'strict', 'ignore' or 'replace'

tail(*file_path*, *lines=10*, *encoding='utf-8'*, *printed=True*, *errors='strict'*)
A really silly way to get the last N lines, defaults to 10.

Parameters

- **file_path** – Path to file to read
- **lines** – Number of lines to read in
- **encoding** – defaults to utf-8 to decode as, will fail on binary
- **printed** – Automatically print the lines instead of returning it
- **errors** – Decoding errors: 'strict', 'ignore' or 'replace'

Returns if printed is false, the lines are returned as a list

cp(*src*, *dst*, *overwrite=False*)
Copy files to a new location.

Parameters

- **src** – list (or string) of paths of files to copy
- **dst** – file or folder to copy item(s) to
- **overwrite** – IF the file already exists, should I overwrite it?

6.9 Wrappers

unique(*max_retries=10*, *wait=0*, *alt_return='-no_alt_return-'*, *exception=<class 'Exception'>*, *error_text=None*)

Wrapper. Makes sure the function's return value has not been returned before or else it run with the same inputs again.

Message format options: {func} {args} {kwargs}

Parameters

- **max_retries** – int of number of retries to attempt before failing
- **wait** – float of seconds to wait between each try, defaults to 0
- **exception** – Exception type of raise
- **error_text** – text of the exception
- **alt_return** – if specified, an exception is not raised on failure, instead the provided value of any type of will be returned

time_it(*log=None*, *message=None*, *append=None*)

Wrapper. Time the amount of time it takes the execution of the function and print it.

If log is true, make sure to set the logging level of 'reusables' to INFO level or lower.

```
import time
import reusables

reusables.add_stream_handler('reusables')

@reusables.time_it(log=True, message="{seconds:.2f} seconds")
def test_time(length):
    time.sleep(length)
    return "slept {0}".format(length)

result = test_time(5)
# 2016-11-09 16:59:39,935 - reusables.wrappers  INFO      5.01 seconds

print(result)
# slept 5
```

Message format options: {func} {seconds} {args} {kwargs}

Parameters

- **log** – log as INFO level instead of printing
- **message** – string to format with total time as the only input
- **append** – list to append item too

catch_it(*exceptions=(<class 'Exception'>,)*, *default=None*, *handler=None*)

If the function encounters an exception, catch it, and return the specified default or sent to a handler function instead.

```
def handle_error(exception, func, *args, **kwargs):
    print(f"{func.__name__} raised {exception} when called with {args}")

@reusables.catch_it(handler=err_func)
```

(continues on next page)

(continued from previous page)

```
def will_raise(message="Hello")
    raise Exception(message)
```

Parameters

- **exceptions** – tuple of exceptions to catch
- **default** – what to return if the exception is caught
- **handler** – function to send exception, func, *args and **kwargs

log_exception(*log='reusables'*, *message=None*, *exceptions=(<class 'Exception'>,)*, *level=40*, *show_traceback=True*)

Wrapper. Log the traceback to any exceptions raised. Possible to raise custom exception.

```
@reusables.log_exception()
def test():
    raise Exception("Bad")

# 2016-12-26 12:38:01,381 - reusables   ERROR  Exception in test - Bad
# Traceback (most recent call last):
#     File "<input>", line 1, in <module>
#     File "reusables\wrappers.py", line 200, in wrapper
#     raise err
# Exception: Bad
```

Message format options: {func} {err} {args} {kwargs}

Parameters

- **exceptions** – types of exceptions to catch
- **log** – log name to use
- **message** – message to use in log
- **level** – logging level
- **show_traceback** – include full traceback or just error message

retry_it(*exceptions=(<class 'Exception'>,)*, *tries=10*, *wait=0*, *handler=None*, *raised_exception=<class 'reusables.shared_variables.ReusablesError'>*, *raised_message=None*)

Retry a function if an exception is raised, or if output_check returns False.

Message format options: {func} {args} {kwargs}

Parameters

- **exceptions** – tuple of exceptions to catch
- **tries** – number of tries to retry the function
- **wait** – time to wait between executions in seconds
- **handler** – function to check if output is valid, must return bool
- **raised_exception** – default is ReusablesError
- **raised_message** – message to pass to raised exception

lock_it(*lock=<unlocked _thread.lock object>*)

Wrapper. Simple wrapper to make sure a function is only run once at a time.

Parameters **lock** – Which lock to use, uses unique default

queue_it(*queue=<queue.Queue object>*, ***put_args*)
Wrapper. Instead of returning the result of the function, add it to a queue.

Parameters **queue** – Queue to add result into

6.10 Numbers

cut(*string*, *characters=2*, *trailing='normal'*)
Split a string into a list of N characters each.

```
reusables.cut("abcdefghi")
# ['ab', 'cd', 'ef', 'gh', 'i']
```

trailing gives you the following options:

- normal: leaves remaining characters in their own last position
- remove: return the list without the remainder characters
- combine: add the remainder characters to the previous set
- error: raise an IndexError if there are remaining characters

```
reusables.cut("abcdefghi", 2, "error")
# Traceback (most recent call last):
#     ...
# IndexError: String of length 9 not divisible by 2 to splice

reusables.cut("abcdefghi", 2, "remove")
# ['ab', 'cd', 'ef', 'gh']

reusables.cut("abcdefghi", 2, "combine")
# ['ab', 'cd', 'ef', 'ghi']
```

Parameters

- **string** – string to modify
- **characters** – how many characters to split it into
- **trailing** – "normal", "remove", "combine", or "error"

Returns list of the cut string

int_to_roman(*integer*)
Convert an integer into a string of roman numbers.

Parameters **integer** –

Returns roman string

int_to_words(*number*, *european=False*)
Converts an integer or float to words.

Parameters

- **number** – String, integer, or float to convert to words. The decimal can only be up to three places long, and max number allowed is 999 decillion.
- **european** – If the string uses the european style formatting, i.e. decimal points instead of commas and commas instead of decimal points, set this parameter to True

Returns The translated string

roman_to_int(*roman_string*)
Converts a string of roman numbers into an integer.

Parameters **roman_string** – XVI or similar

Returns parsed integer

6.11 Changelog

6.11.1 Version 0.9.3

- Fixing imports in python 3.7

6.11.2 Version 0.9.2

- Adding option of kwargs to task for tasker
- Fixing documentations links

6.11.3 Version 0.9.1

- Fixing local imports not working when installed

6.11.4 Version 0.9.0

- Adding datetime_format, dtf methods
- Adding datetime_from_iso, dtiso methods
- Adding catch_it and retry_it wrappers
- Adding CONTRIBUTING file
- Changing Namespace now operates more like "dict" on init, and can accept both iterable and kwargs
- Changing major structure of reusables to better group similar functionality
- Changing wrapper time_it now uses .time for older versions instead of the .clock
- Depreciation Warning: get_logger is changing to setup_logger
- Breaking change: log_exception has new and changed kwargs
- Breaking change: removing Cookie Management in favor of separate library
- Breaking change: removing sort_by
- Breaking change: removing namespace.from_dict()
- Breaking change: removing DateTime class in favor of singular methods datetime_format and datetime_from_iso

6.11.5 Version 0.8.0

- Adding log_exception wrapper
- Adding ProtectedDict
- Adding hooks for Tasker main loop
- Adding roman number functions
- Adding directory_duplicates function
- Adding integer to words functions
- Adding option to enable scandir package walk instead of os.walk
- Adding url_to_ip and ip_to_url functions
- Adding hex_digest kwarg to file_hash
- Adding args and kwargs to time_it and log_exception wrappers
- Fixing file_hash checks by just passing to hashlib
- Changing functions to remove 'all' from them, extract_all, archive_all, find_all_files and find_all_files_generator
- Changing time_it function to use time.perf_counter on 3.3+ and time.clock on 2.6-3.2
- Depreciation Warning: extract_all is changing to extract
- Depreciation Warning: archive_all is changing to archive
- Depreciation Warning: find_all_files is changing to find_files
- Depreciation Warning: find_all_files_generator is changing to find_files_generator
- Depreciation Warning: count_all_files is changing to count_files
- Breaking change: Removing reuse wrapper
- Breaking change: archive_all now detects type based off name, should supply extension to name
- Breaking change: time_it message now takes args seconds, args, kwargs and does not allow positionals
- Breaking change: os_tree will no longer return an empty dictionary on failure, but include the base directory supplied
- Breaking change: renaming splice to cut

6.11.6 Version 0.7.0

- Adding archive_all and now methods
- Adding logger helpers to add stream and file handlers
- Adding depth and abspath to find files methods
- Adding head, tail, cat bash equivalents
- Adding command queue to Tasking class, to give commands asynchronously and without directly referencing the instance
- Changing test suite to have a common file it pulls imports and info from
- Changing logger helpers to accept string instead of logger

- Breaking change: Moving log formats from variables to Namespace log_formats
- Breaking change: Moving command line helpers to cli
- Breaking change: Command line helpers are not imported by default, should now use: from reusables.cli import *
- Breaking change: join_root has been better named join_here

6.11.7 Version 0.6.1

- Changing config_dict auto_find to accept a path to search at
- PyPI is stupid is why 0.6.0 is not up there

6.11.8 Version 0.6.0

- Adding multiprocessing helpers, Tasker class and run_in_pool
- Adding download and cmd helper functions
- Adding ThreadedServer class, for running a server (defaults to local file server) in the background
- Adding terminal analogue functions: cd, pwd, ls, pushd, popd
- Adding match_case option for find_all_files and count_all_files
- Fix 'run' call to CalledProcessError on older python versions
- Changing logger to _logger to be hidden by default (should not be breaking, if so you coded wrong)

6.11.9 Version 0.5.2

- Fix setup.py to use __init__.py instead of reusables.py for attrs

6.11.10 Version 0.5.1

- Adding default argument to configNamespace's int, float, list and boolean methods
- Adding change_logger_levels
- Changing __version__ location to __init__ so it can be accessed properly
- Changing protected_keys in Namespace to be hidden from documentation
- Changing linux only tests to be in their own class
- Breaking change: keyword arg position for configNamespace.list now has 'default' as first kwarg

6.11.11 Version 0.5.0

- Adding ConfigNamespace
- Adding lock wrapper for functions
- Adding duplicate file finder
- Adding easy CSV / list transformation

- Adding protected keys for namespaces
- Adding touch
- Adding extensions for scripts, binary and markup files
- Changing logging to be more explicit and run on sys.stdout
- Breaking change: removed command line running options and main function

6.11.12 Version 0.4.1

- Fixing Firefox dump command not working
- Adding MissingCookiesDB exception for clearer
- Wrapping commits with exceptions clauses for BrowserException
- Adding "created" and "expires" in _row_to_dict for Browsers

6.11.13 Version 0.4.0

- Breaking change: Removed 'dnd' from functions for clearer 'dry_run' or 'delete_on_success'
- Breaking change: Removing 'dangerzone' file, moving 'reuse' function to root namespace
- Added management tools for Firefox and Chrome cookies
- Added unique wrapper tool, ensures return value has not been returned before
- Changed all top level imports to have underscore before them like standard library

6.11.14 Version 0.3.0

- Namespace re-written to be more compatible with built-in dict
- Added support for rarfile extraction
- Adding PY2, PY3 as compliments of the booleans python3x to be similar to the six package
- Adding logging features
- Separating functionality to individual files
- Adding sphinx generated API documentation

6.11.15 Version 0.2.0

- Added DateTime class
- Added and rearranged several regular expression
- Added tree_view of dictionaries
- Added os_tree of a directory to a dictionary

6.11.16 Version 0.1.3

- Addition of Makefile
- Fixing issues with setup.py not including all needed files
- Tests now pass on windows by skipping some linux specific tests
- Improved config tests to only test against known sections, instead of entire dictionaries

6.11.17 Version 0.1.2

- Name change from reuse to reusables due to name already being registration on pypi

6.11.18 Version 0.1.1

- find_all_files_iter renamed to find_all_files_generator
- Added python2.6 and pypy testing and support
- Namespace is now a subclass of dict.
- Changing Readme into rst format.

6.11.19 Version 0.1

- initial release

Python Module Index

r

Index

L

M

N

O

P

Q

R

S

T

U