rdial

Release 0.15.0

Apr 02, 2017

Contents

`rdial` is a simple way to track the time you spend on tasks. It tracks the name of a task, its start time, its duration and optionally a message... nothing more.

It is written in Python, and requires v2.6 or later. *`rdial`* is released under the GPL v3.

Git repository https://github.com/JNRowe/rdial/

Issue tracker https://github.com/JNRowe/rdial/issues/

Contributors https://github.com/JNRowe/rdial/contributors/

CHAPTER 1

Contents

Background

I spend an awful lot of time sitting in front of a computer, working on a huge number of disparate projects. And when it comes time to gauge what I've been working on it would be nice if I could just fire up a simple tool to see where I've been spending my time.

The features *I* need in a time tracking tool are:

- Easily accessible data, in a format that can be processed simply
- Ability to work off-line, because those times are considerably more common than some people seem to think
- Works on all the platforms I regularly use; desktop, mobile phone, ZipIt, and more

Now `rdial` is born, and I should be able to realise those dreams!

Philosophy

There are a few interface choices in **`rdial`** that may be need a little explanation. They may well be enough to deter you from using **`rdial`** at all, but that is fine as there are *plenty more options* out there!

Explicit new tasks

> It is an error to attempt to start a task that doesn't already exist in the database

If you wish to create a new task you *must* give the `--new` option when starting the task. This should hopefully catch typos and task name "thinkos", and it has proven to do so for me.

Switch vs "stopstart"

> It is an error to attempt to start a task when another is running, or switch a task when one is not running

If you wish to start a new task you *must* stop the previous one. At first it seems natural to just accept that **rdial start** should complete the previous task, but doing so encourages users to not be aware of their current state.

Similarly, if you wish to switch to a new task then a task *must* be running. It might be convenient to make **rdial switch** just start the new task if one is not running, but again it encourages users to be unaware of their current state.

Usage

The **rdial** script is the main workhorse of *rdial*.

See *Getting started* for basic usage examples.

Options

--version
Show the version and exit.

-d <directory>, **--directory**=<directory>
Database location, defaults to `${XDG_DATA_HOME:-~/.local/share}/rdial`.

--backup/--no-backup
Write data file backups.

--cache/--no-cache
Do not write cache files.

--config <file>
File to read configuration data from, defaults to `${XDG_CONFIG_HOME:-~/.config}/rdial/config`.

-i, --interactive/--no-interactive
Support interactive message editing.

--help
Show help message and exit.

Commands

fsck - Check storage consistency

```
rdial fsck [--help]
```

--help
Show help message and exit.

start - Start task

```
rdial start [--help] [-x] [-n] [-t time] <task>
```

-x, --from-dir
Use directory name as task name.

-n, --new
Start a new task.

-t <time>, **--time** <time>
Manually set start time for task.

--help
Show help message and exit.

`stop` - Stop task

```
rdial stop [--help] [-m <message>] [--amend]
```

-m <message>, **--message**=<message>
Closing message.

-F <file>, **--file** <file>
Read closing message from file.

--amend
Amend previous stop entry.

--help
Show help message and exit.

`switch` - Switch to another task

```
rdial switch [--help] [-x] [-n] [-m <message>] [task]
```

-x, --from-dir
Use directory name as task name.

-n, --new
Start a new task.

-t <time>, **--time** <time>
Manually set start time for task.

-m <message>, **--message** <message>
Closing message for current task.

-F <file>, **--file** <file>
Read closing message for current task from file.

--help
Show help message and exit.

`run` - Run command with timer

```
rdial run [--help] [-x] [-n] [-t time] [-m message] [-F file] [-c command] <task>
```

-x, --from-dir
Use directory name as task name.

-n, --new
Start a new task.

-t <time>, **--time** <time>
Manually set start time for task.

-m <message>, **--message** <message>
Closing message for current task.

-F <file>, **--file** <file>
Read closing message for current task from file.

-c <command>, **--command** <command>
Command to run.

--help
Show help message and exit.

`wrapper` - Run predefined command with timer

```
rdial wrapper [--help] [-t time] [-m message] [-F file] <wrapper>
```

See *run wrappers configuration*.

-t <time>, **--time** <time>
Manually set start time for task.

-m <message>, **--message** <message>
Closing message for current task.

-F <file>, **--file** <file>
Read closing message for current task from file.

--help
Show help message and exit.

`report` - Report time tracking data

```
rdial report [--help] [-d <duration>] [-s <order] [-r] [--style] [--stats] <task>
```

-d <duration>, **--duration**=<duration>
Filter events for specified time period {day,week,month,year,all}.

-s <order>, **--sort**=<order>
Field to sort by {task,time}.

-r, **--reverse**
Reverse sort order.

--style
Table output style {grid,latex,mediawiki,orgtbl,pipe,plain,rst,simple,tsv}

--stats
Display database statistics.

-x, **--from-dir**
Use directory name as task name.

--help
Show help message and exit.

`running` - Display running task, if any

```
rdial running [--help]
```

--help
Show help message and exit.

`last` - Display last task, if any

```
rdial last [--help]
```

--help
Show help message and exit.

`ledger` - Generate `ledger` compatible data file

```
rdial ledger [--help] [-d <duration>] [-r RATE] [task]
```

-d <duration>, **--duration**=<duration>
Filter events for specified time period {day,week,month,year,all}.

-r <rate>, **--rate** <rate>
Hourly rate for task output.

-x, --from-dir
Use directory name as task name.

--help
Show help message and exit.

Getting started

Basic usage

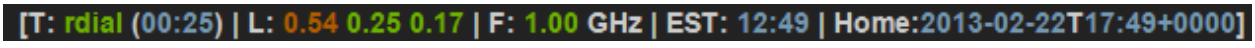
The command interface is hopefully quite intuitive. The following is a sample session:

```
$ rdial start my_task
$ rdial running
Task my_task has been running for 0:12:38
$ rdial stop -m'Fixed bug #40'
Task my_task running for 0:44:00
```

Help on individual subcommands is available via `rdial <subcommand> --help` or in the *Usage* document.

Current task

The current task name is written to the database directory in the `.current` file. You can use its contents to populate notifiers in task bars.



As the file is created when the user executes **`rdial start`** you can also use its modification time to quickly calculate a running time for the task.

See the *Integrating with taskbars* document for some guidance on using `rdial` in various environments.

rdial

Simple time tracking for simple people

SYNOPSIS

rdial [option]... <command>

DESCRIPTION

rdial is a simple way to track the time you spend on tasks. It tracks the name of a task, its start time and its duration... nothing more.

OPTIONS

--version Show the version and exit.

-d <directory>, --directory=<directory> Database location, defaults to `${XDG_DATA_HOME:-~/.local/share}/rdial`.

–backup/–no-backup Write data file backups.

–cache/–no-cache Do not write cache files.

--config <file> File to read configuration data from, defaults to `${XDG_CONFIG_HOME:-~/.config}/rdial/config`.

-i, –interactive/–no-interactive Support interactive message editing.

--help Show help message and exit.

COMMANDS

fsck

Check storage consistency

--help Show help message and exit.

start

Start task

-x, --from-dir Use directory name as task name.

-n, --new Start a new task.

-t <time>, --time <time> Manually set start time for task.

--help Show help message and exit.

stop

Stop task

-m <message>, --message=<message> Closing message.

-F <file>, --file <file> Read closing message from file.

--amend Amend previous stop entry.

--help Show help message and exit.

switch

Switch to another task

-x, --from-dir Use directory name as task name.

-n, --new Start a new task.

-t <time>, --time <time> Manually set start time for task.

-m <message>, --message <message> Closing message for current task.

-F <file>, --file <file> Read closing message for current task from file.

--help Show help message and exit.

run

Run command with timer

-x, --from-dir Use directory name as task name.

-n, --new Start a new task.

-t <time>, --time <time> Manually set start time for task.

-m <message>, --message <message> Closing message for current task.

-F <file>, --file <file> Read closing message for current task from file.

-c <command>, --command <command> Command to run.

--help Show help message and exit.

wrapper

Run predefined command with timer

-t <time>, --time <time> Manually set start time for task.

-m <message>, --message <message> Closing message for current task.

-F <file>, --file <file> Read closing message for current task from file.

--help Show help message and exit.

report

Report time tracking data

-d <duration>, --duration=<duration> Filter events for specified time period {day,week,month,year,all}.

-s <order>, --sort=<order> Field to sort by {task,time}.

-r, --reverse Reverse sort order.

--style Table output style {grid,latex,mediawiki,orgtbl,pipe,plain,rst,simple,tsv}.

--stats Display database statistics.

-x, --from-dir Use directory name as task name.

--help Show help message and exit.

running

Display running task, if any

--help Show help message and exit.

last

Display last task, if any

--help Show help message and exit.

ledger

Generate ledger compatible data file

-d <duration>, --duration=<duration> Filter events for specified time period {day,week,month,year,all}.

-r <rate>, --rate <rate> Hourly rate for task output.

-x, --from-dir Use directory name as task name.

--help Show help message and exit.

BUGS

None known.

AUTHOR

Written by James Rowe

RESOURCES

Home page, containing full documentation: http://rdial.rtfd.org/

Issue tracker: https://github.com/JNRowe/rdial/issues/

COPYING

Configuration

rdial can be configured using a cascading series of files, processed in the following order:

- The package's `rdial/config` file which contains the base configuration
- Any `rdial/config` file that exists in `$XDG_CONFIG_DIRS`
- The user's `rdial/config` file found in `$XDG_CONFIG_HOME`
- The `.rdialrc` found in the current directory

Note: See the XDG base directory specification for more information on using `$XDG_CONFIG_DIRS` and `$XDG_CONFIG_HOME`.

File format

The configuration file is a `INI` format file. There is a section labelled `rdial` for global options, and a section for each subcommand. Each section consists of a series of `name=value` option pairs.

An example configuration file is below:

```
[rdial]
colour = False

[report]
sort = time
reverse = True
```

The configuration files are processed using configobj, and you can make use of all the features it provides(such as interpolation).

`rdial` section

`backup` (default: `True`)

If this key is set to `True` then backup data files are written with a ~ suffix.

Warning: You are strongly urged to keep this set to `True`, as it helps to protect you from bugs in *rdial*.

`colour` (default: `True`)

If this key is set to `False` then no coloured output will be produced by `rdial`.

The key `color` is also accepted.

`directory` (default: *`$XDG_DATA_HOME`*`/rdial`)

This key sets the location of your data files. Some users use this, combined with the per-directory config file, to keep per-project task databases.

`interactive` (default: `False`)

If this key is set to `True` then `rdial` will interactively ask the user for for messages if they're not supplied as arguments.

`run wrappers` section

This section is used to configure pre-defined arguments for the *rdial run* subcommand. It consists of a series of string keys to use as the wrapper title, and arguments to the **`rdial run`** subcommand as values. For example:

```
[run wrappers]
feeds = -c 'mutt -f ~/Mail/RSS2email/' procast
calendar = -c 'wyrd ~/.reminders/events' calendar
```

The above configuration entry `feeds` allows us to use `rdial wrapper` *`feeds`* to open `mutt` in a specific mailbox, and time our usage under the `procast` task.

Configuration via environment variables

`rdial` defaults for many options can be configured via environment variables. The design purpose for supporting these environment variables is to make it easy for users to configure per-project defaults using shell hooks.

`RDIAL_BACKUP`
: This controls whether **`rdial`** creates backup of data files. It must be a boolean setting that accepts `false`/`true`, `0`/`1` or `y`/`n` as its value.

`RDIAL_CACHE`
: This controls whether **`rdial`** creates a cache for data files. It must be a boolean setting that accepts `false`/`true`, `0`/`1` or `y`/`n` as its value.

`RDIAL_COLOUR`
: This controls whether **`rdial`** displays informational messages in colour. It must be a boolean setting that accepts `false`/`true`, `0`/`1` or `y`/`n` as its value.

`RDIAL_CONFIG`
: The location of the *configuration* file. It must be a string value.

`RDIAL_DIRECTORY`
: The location of the **`rdial`** storage directory. It must be a string value.

`RDIAL_INTERACTIVE`
: This controls whether **`rdial`** asks for messages interactively if they're not provided as arguments. It must be a boolean setting that accepts `false`/`true`, `0`/`1` or `y`/`n` as its value.

RDIAL_RATE
The value for the *rdial ledger -r* hourly rate setting. It must be a numeric value.

RDIAL_REVERSE
This controls whether **rdial** inverts the sort order when displaying reports. It is a boolean setting that accepts false/true, 0/1 or y/n as its value.

RDIAL_SORT
This controls the sorting order for reports generated by **rdial**. It can either task or time to sort by task name or cumulative time.

RDIAL_TASK
This controls the default task name for **rdial**, and is a good way to configure a project default within a shell hook. It must be a string value.

Frequently Asked Questions

Why must I specify --new when creating a new task

Perhaps you're super special and never produce a typo, I'm not. The --new option to **rdial start** is a fix for this common – to me – problem.

The **rdial switch** command is of related design. You can not use it when no task is running, and attempting to do so is probably a sign you've lost track of your state. I consider this to be a massive problem, and would rather not sidestep it by allowing **rdial start** to stop running tasks or **rdial switch** to work without a running task.

Where does the name rdial come from?

Around the beginning of 2012 I wrote a very simple shell script to track my time, and at some point I decided it should be safer and cleaner. I came up with a very clever and imaginative name for this new project, and then promptly forgot how it came about.

Since then I haven't even managed to come up with a clever backronym for it.

How do I process rdial data files using go

If you keep receiving ErrTrailingComma errors when reading files using golang it is because you are processing files with empty message fields. The default behaviour of the encoding/csv pkg is to raise an error when it encounters an empty final field. You can tell go to accept empty final fields by setting the TrailingComma attribute on your CSV reader.

```
reader := csv.NewReader(file)
reader.TrailingComma = true
```

Integrating with taskbars

The following sections should give you some idea of how you can (ab)use the `.current` file in your window manager of choice.

Note: There are exactly two examples right now, because these are the two environments I use. Feel free to submit your own!

Todo

Add more fleshed out examples.

---

awesome

For example, with awesome, you could create a simple timer based widget that shows the currently running task:

```
GLib = lgi.GLib
tasktext = wibox.widget.textbox!
tasktimer = with timer timeout: 30
    \connect_signal "timeout", ->
        if file = io.open GLib.get_user_data_dir! .. "/rdial/.current"
            tasktext\set_markup file\read!
            file\close!
        else
            tasktext\set_markup "none"
    -- fire timer for initial update
    \emit_signal "timeout"
    \start!
```

Note: The above example is compact but very naïve, and will be incorrect in the time between state changes and updates. If you're implementing your own widget you'll be better served by using GFileMonitor to track state changes.

You could also hook the `mouse::enter` and `mouse::leave` signals to create a naughty popup showing the task time, or use awful.button to allow you to switch tasks directly from the taskbar.

dwm

With dwm you're basically free to pump the status bar however you wish. You could, for example, just show the current task with the following genie snippet:

```
[indent=4]

uses
    X
    Posix

init
    var file = GLib.Environment.get_user_data_dir() + "/rdial/.current"
    var dpy = new X.Display()
    var root = dpy.default_root_window()
```

```
    text : string

    while true
        if GLib.FileUtils.test(file, GLib.FileTest.IS_REGULAR)
            GLib.FileUtils.get_contents(file, out text)
        else
            text = "none"
        dpy->change_property(root, XA_WM_NAME, XA_STRING, 8,
                             PropMode.Replace, (array of uchar)text,
                             text.length)
        dpy->flush()
        Posix.sleep(30)
```

Note: The above example is compact but very naïve, and will be incorrect in the time between state changes and updates. If you're implementing your own status tool you'll be better served by using GFileMonitor to track state changes.

You could also implement a simple task manager using dmenu, the following zsh snippet shows how to build a selector for an existing task:

```
echo ${XDG_DATA_HOME:-~/.local/share}/rdial/*~*~(:t:s/.csv/) |
    tr ' ' '\n' |
    dmenu -p "task?"
```

Alternatives

Before diving in and spitting out this package I looked at the alternatives below. If I have missed something please drop me a mail.

It isn't meant to be unbiased, and you should try the packages out for yourself. I keep it here mostly as a reference for myself, and maybe to help out people who are already familiar with one of the entries below so they can see where I'm coming from.

Todo

Check for recent additions to arena

gtimelog

`gtimelog` is an interesting tool, and ticks most of the boxes for me. The ideas are quite well thought out, and the interface is very simple to use. I'm sure it's great for people with strictly structured working days, but that definitely isn't me.

As a side note, when we were playing with it at the office a number of co-workers stumbled across various bugs that dogged its usage. Unfortunately, the project is developed with bzr and launchpad, so it was simply abandoned in favour of trying to fix it. Most moved on to `org-mode`, and some to `rdial`.

hammertime

hammertime is a great tool for tracking time in a simple manner, however it has a couple of drawbacks for my use case.

Firstly, it stores data in a `git` branch which means all projects need their own `git` repository. This works surprisingly well for the most part, but makes fetching all the stored data across multiple projects quite cumbersome.

The more significant problem *for me* is that the implementation works by stashing changes and switching branches, which will cause annoying rebuilds every time you call `git time`. This could be fixed however by using `git hash-object` directly for storing updates and `git cat-file` for reading data, should anyone be interested in working on it.

I still happily recommend it to people who are simply trying to log the time they spend working on small projects.

ktimetracker

Works well, but isn't available on most of the platforms I care about. If KDE is available everywhere you care about, I'd heartily recommend it.

org-mode

org-mode includes fantastic time tracking support, and some excellent reporting mechanisms. You can interleave your time tracking with other data, maintain hierarchies of projects with their own time tracking data and take advantage of all the other features `org-mode` has to offer.

If I had a copy of emacs available everywhere I wanted to log time data I wouldn't even consider using anything else. And if you use emacs then you shouldn't either!

taskwarrior

There is some support for time tracking in taskwarrior, and if you only need to maintain a log of the time you spend on previously defined tasks it is probably enough to get by.

I still take advantage of its functionality now, in combination with `rdial`, so that I can see when I started working toward a task in my to-do list.

Release HOWTO

Test

In the general case tests can be run via `nose2`:

```
$ nose2 -vv tests
```

When preparing a release it is important to check that *rdial* works with all supported Python versions, and that the documentation is correct.

Prepare release

With the tests passing, perform the following steps

- Update the version data in `rdial/_version.py`
- Update `NEWS.rst`, if there are any user visible changes
- Commit the release notes and version changes
- Create a signed tag for the release
- Push the changes, including the new tag, to the GitHub repository
- Create new release on GitHub

Update PyPI

Create and upload the new release tarballs to PyPI:

```
$ ./setup.py sdist bdist_wheel register upload --sign
```

Fetch the uploaded tarballs, and check for errors.

You should also perform test installations from PyPI, to check the experience *rdial* users will have.

Todo items for documentation

Todo

Check for recent additions to arena

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/rdial/checkouts/latest/doc/alternatives.rst, line 14.)

Todo

Add more fleshed out examples.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/rdial/checkouts/latest/doc/taskbars.rst, line 15.)

API documentation

Note: The documentation in this section is aimed at people wishing to contribute to *rdial*, and can be skipped if you are simply using the tool from the command line.

Event

Note: The documentation in this section is aimed at people wishing to contribute to *rdial*, and can be skipped if you are simply using the tool from the command line.

class rdial.events.**Event**(*task*, *start=None*, *delta=None*, *message=''*)
Initialise a new Event object.

Parameters

- **task** (*str*) – Task name to tracking
- **start** (*datetime.datetime*) – Start time for event
- **delta** (*datetime.timedelta*) – Duration for event
- **message** (*str*) – Message to attach to event

running()
Check if event is running.

Returns Event name, if running

Return type str

stop(*message=None*, *force=False*)
Stop running event.

Parameters

- **message** (*str*) – Message to attach to event
- **force** (*bool*) – Re-stop a previously stopped event

Raises *TaskNotRunningError* – Event not running

writer()
Prepare object for export.

Returns Event data for object storage

Return type dict

class rdial.events.**Events**(*iterable=None*, *backup=True*)
Initialise a new Events object.

Parameters

- **iterable** (*list*) – Objects to add to container
- **backup** (*bool*) – Whether to create backup files

context(*directory*, *backup=True*, *write_cache=True*)
Convenience context handler to manage reading and writing database.

Warning: Deprecated name for wrapping

dirty
Modified tasks requiring sync against storage.

filter(*filt*)
Apply filter to events.

Parameters **filt** (*types.FunctionType*) – Function to filter with

Returns Events matching given filter function

Return type *Events*

for_date(*year*, *month=None*, *day=None*)
Filter events for a specific date.

Parameters

- **year** (*int*) – Year to filter on
- **month** (*int*) – Month to filter on, or None
- **day** (*int*) – Day to filter on, or None

Returns Events occurring within specified date

Return type *Events*

for_task(*task*)
Filter events for a specific task.

Parameters **task** (*str*) – Task name to filter on

Returns Events marked with given task name

Return type *Events*

for_week(*year*, *week*)
Filter events for a specific ISO-8601 week.

Parameters

- **year** (*int*) – Year to filter events on
- **week** (*int*) – ISO-8601 month number to filter events on

Returns Events occurring in given ISO-8601 week

Return type *Events*

last()
Return current/last event.

This handles the empty database case by returning None

Returns Last recorded event

Return type *Event*

static **read**(*directory*, *backup=True*, *write_cache=True*)
Read and parse database.

Assume a new *Events* object should be created if the file is missing

Parameters

- **directory** (*str*) – Location to read database files from
- **backup** (*bool*) – Whether to create backup files
- **write_cache** (*bool*) – Whether to write cache files

Returns Parsed events database

Return type *Events*

running()
Check if an event is running.

We return the running task, if one exists, for easy access.

Returns Running event, if an event running

Return type *Event*

start(*task*, *new=False*, *start=''*)
Start a new event.

Parameters

- **task** (*str*) – Task name to tracking
- **start** (*str*) – ISO-8601 start time for event

Raises *TaskRunningError* – An event is already running

stop(*message=None*, *force=False*)
Stop running event.

Parameters

- **message** (*str*) – Message to attach to event
- **force** (*bool*) – Re-stop a previously stopped event

Raises *TaskNotRunningError* – No task running!

sum()
Sum duration of all events.

Returns Sum of all event deltas

Return type datetime.timedelta

tasks()
Generate a list of tasks in the database.

Returns Names of tasks in database

Return type list of str

static wrapping(**args*, ***kwds*)
Convenience context handler to manage reading and writing database.

Parameters

- **directory** (*str*) – Database location
- **backup** (*bool*) – Whether to create backup files
- **write_cache** (*bool*) – Whether to write cache files

write(*directory*)
Write database file.

Parameters **directory** (*str*) – Location to write database files to

class rdial.events.**RdialDialect**
CSV dialect for rdial data files.

Examples

```
>>> event = Event('test')
>>> event.running()
'test'
>>> event.stop('complete')
>>> event.running()
False
>>> data = event.writer()
>>> data['message']
'complete'
>>> event2 = Event('test', start="2013-01-01T12:00:00Z")

>>> events = Events([event, event2])
>>> events.filter(lambda x: x.message == 'complete')
Events([Event('test', '...', '...', 'complete')])
>>> events.for_date(2013, 1)
Events([Event('test', '2013-01-01T12:00:00Z', '', '')])
>>> events.for_week(2012, 53)
Events([Event('test', '2013-01-01T12:00:00Z', '', '')])
>>> events.running()
'test'
```

Command line

Note: The documentation in this section is aimed at people wishing to contribute to *rdial*, and can be skipped if you are simply using the tool from the command line.

Commands

rdial.cmdline.**bug_data**()
Produce data for rdial bug reports.

rdial.cmdline.**fsck**()
Check storage consistency.

Parameters

- **ctx** (*click.Context*) – Current command context
- **globs** (*utils.AttrDict*) – Global options object

rdial.cmdline.**start**()
Start task.

Parameters

- **globs** (*utils.AttrDict*) – Global options object
- **task** (*str*) – Task name to operate on
- **new** (*bool*) – Create a new task
- **time** (*datetime.datetime*) – Task start time

rdial.cmdline.**stop**()
Stop task.

Parameters

- **globs** (*utils.AttrDict*) – Global options object
- **message** (*str*) – Message to assign to event
- **fname** (*str*) – Filename to read message from
- **amend** (*bool*) – Amend a previously stopped event

rdial.cmdline.**switch**()
Complete last task and start new one.

Parameters

- **globs** (*utils.AttrDict*) – Global options object
- **task** (*str*) – Task name to operate on
- **new** (*bool*) – Create a new task
- **time** (*datetime.datetime*) – Task start time
- **message** (*str*) – Message to assign to event
- **fname** (*str*) – Filename to read message from

rdial.cmdline.**run**()
Run timed command.

Parameters

- **globs** (*utils.AttrDict*) – Global options object
- **task** (*str*) – Task name to operate on
- **new** (*bool*) – Create a new task
- **time** (*datetime.datetime*) – Task start time
- **message** (*str*) – Message to assign to event
- **fname** (*str*) – Filename to read message from
- **command** (*str*) – Command to run

rdial.cmdline.**wrapper**()
Run predefined timed command.

Parameters

- **ctx** (*click.Context*) – Click context object
- **globs** (*utils.AttrDict*) – Global options object
- **time** (*datetime.datetime*) – Task start time
- **message** (*str*) – Message to assign to event
- **fname** (*str*) – Filename to read message from
- **wrapper** (*str*) – Run wrapper to execute

rdial.cmdline.**report**()
Report time tracking data.

Parameters

- **globs** (*utils.AttrDict*) – Global options object

- **task** (*str*) – Task name to operate on
- **stats** (*bool*) – Display short overview of data
- **duration** (*str*) – Time window to filter on
- **sort** (*str*) – Key to sort events on
- **reverse** (*bool*) – Reverse sort order
- **style** (*str*) – Table formatting style

rdial.cmdline.**running**()
Display running task, if any.

Parameters **globs** (*utils.AttrDict*) – Global options object

rdial.cmdline.**last**()
Display last event, if any.

Parameters **globs** (*utils.AttrDict*) – Global options object

rdial.cmdline.**ledger**()
Generate ledger compatible data file.

Parameters

- **globs** (*utils.AttrDict*) – Global options object
- **task** (*str*) – Task name to operate on
- **duration** (*str*) – Time window to filter on
- **rate** (*str*) – Rate to assign hours in report

Entry points

rdial.cmdline.**cli**()
Main command entry point.

Parameters

- **ctx** (*click.Context*) – Current command context
- **directory** (*str*) – Location to store event data
- **backup** (*bool*) – Whether to create backup files
- **cache** (*bool*) – Whether to create cache files
- **config** (*str*) – Location of config file
- **interactive** (*bool*) – Whether to support interactive message editing

rdial.cmdline.**main**()
Command entry point to handle errors.

Returns Final exit code

Return type int

Command support

rdial.cmdline.**filter_events**(*globs*, *task=None*, *duration=None*)
Filter events for report processing.

Parameters

- **globs** (*utils.AttrDict*) – Global options object
- **task** (*str*) – Task name to filter on
- **duration** (*str*) – Time window to filter on

Returns Events matching specified criteria

Return type *Events*

rdial.cmdline.**get_stop_message**(*current*, *edit=False*)
Interactively fetch stop message.

Parameters

- **current** (events.Event) – Current task
- **edit** (*bool*) – Whether to edit existing message

Returns Message to use

Return type str

CLI support

class rdial.cmdline.**HiddenGroup**(*name=None*, *commands=None*, ***attrs*)
Support for 'hidden' commands.

Any click.Command with the hidden attribute set will not be visible in help output. This is mainly to be used for diagnostic commands, and shouldn't be abused!

class rdial.cmdline.**TaskNameParamType**
Task name parameter handler.

convert(*value*, *param*, *ctx*)
Check given task name is valid.

Parameters

- **value** (*str*) – Value given to flag
- **param** (*click.Argument*) – Parameter being processed
- **ctx** (*click.Context*) – Current command context

Returns Valid task name

Return type str

class rdial.cmdline.**StartTimeParamType**
Start time parameter handler.

convert(*value*, *param*, *ctx*)
Check given start time is valid.

Parameters

- **value** (*str*) – Value given to flag

- **param** (*click.Argument*) – Parameter being processed
- **ctx** (*click.Context*) – Current command context

Returns Valid start time

Return type datetime.datetime

rdial.cmdline.**task_from_dir**(*ctx*, *param*, *value*)
Override task name default using name of current directory.

Parameters

- **ctx** (*click.Context*) – Current command context
- **param** (*click.Argument*) – Parameter being processed
- **value** (*bool*) – True if flag given

rdial.cmdline.**task_option**(*fun*)
Add task selection options.

Parameters **fun** (*types.FunctionType*) – Function to add options to

Returns Function with additional options

Return type types.FunctionType

rdial.cmdline.**duration_option**(*fun*)
Add duration selection option.

Note: This is only here to reduce duplication in command setup.

Parameters **fun** (*types.FunctionType*) – Function to add options to

Returns Function with additional options

Return type types.FunctionType

rdial.cmdline.**message_option**(*fun*)
Add message setting options.

Parameters **fun** (*types.FunctionType*) – Function to add options to

Returns Function with additional options

Return type types.FunctionType

Utilities

Note: The documentation in this section is aimed at people wishing to contribute to *rdial*, and can be skipped if you are simply using the tool from the command line.

Convenience functions and classes

rdial.utils.**write_current**(*fun*)
Decorator to write `current` file on function exit.

See also:

Integrating with taskbars

Returns Wrapped function

Return type types.FunctionType

rdial.utils.**remove_current**(*fun*)
Decorator to remove `current` file on function exit.

See also:

Integrating with taskbars

Returns Wrapped function

Return type types.FunctionType

rdial.utils.**newer**(*fname*, *reference*)
Check whether given file is newer than reference file.

Parameters

- **fname** (*str*) – File to check
- **reference** (*str*) – file to test against

Returns True if `fname` is newer than `reference`

Return type bool

Time handling

rdial.utils.**parse_datetime_user**(*string*)
Parse datetime string from user.

We accept the normal ISO-8601 formats, but kick through to the formats supported by the system's date command if parsing fails.

Parameters **string** (*str*) – Datetime string to parse

Returns Parsed datetime object

Return type datetime.datetime

Examples

Time handling

```
>>> parse_datetime_user('40 minutes ago')
datetime.datetime(2012, 2, 15, 18, 59, 18, tzinfo=<UTC>
```

Errors

Note: The documentation in this section is aimed at people wishing to contribute to *rdial*, and can be skipped if you are simply using the tool from the command line.

exception rdial.utils.**RdialError**
Generic exception for rdial.

exception rdial.events.**TaskNotRunningError**
Exception for calling mutators when a task is not running.

exception rdial.events.**TaskRunningError**
Exception for starting task when a task is already running.

Examples

```
>>> events_stopped.stop()
Traceback (most recent call last):
  ...
TaskNotRunningError: No task running!
>>> events_running.start('rdial', new=True)
Traceback (most recent call last):
  ...
TaskRunningError: Running task test!
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

Symbols

B

C

S

T

W