rdial

Release 1.2.1

James Rowe

Jun 26, 2019

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 v3.6 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

ONE

CONTENTS

1.1 Background

I spend an awful lot of time sitting in front of a computer, working on many disparate projects. When I need to gauge my time it would be great 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 can realise those dreams!

1.1.1 Philosophy

A few interface choices in **rdial** may be need a little explanation. Any one of them may well be enough to deter you from using **rdial** at all, but that is fine as *other options* are out there!

Explicit new tasks

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

If you wish to create a new task you *must* give the `rdial start --new` option when starting the task. This should — with hope — catch typos and task name "thinkos", and it has proven to do exactly that for me.

Switch vs "stopstart"

> It is an error to try to start a task when another is running, or switch a task when no task is running

If you wish to start a new task you *must* stop the previous task. 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 a task is not running, but again it encourages users to be unaware of their current state.

1.2 Usage

1.2.1 rdial

Minimal time tracking for maximal benefit.

```
rdial [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

-d, --directory <directory>
Directory to read/write to.

--backup, --no-backup
Do not write data file backups.

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

--config <config>
File to read configuration data from.

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

--colour, --no-colour
Output colourised informational text.

Environment variables

RDIAL_COLOUR

Provide a default for *--colour*

bug-data

fsck

Check storage consistency.

```
rdial fsck [OPTIONS]
```

Options

-p, --progress, -q, --no-progress
Display progress bar.

last

Display last event, if any.

```
rdial last [OPTIONS]
```

ledger

Generate ledger compatible data file.

```
rdial ledger [OPTIONS] [TASK]
```

Options

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

-d, --duration <duration>
Filter events for specified time period.

Options day|week|month|year|all

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

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

RDIAL_RATE

Provide a default for *-r*

report

Report time tracking data.

```
rdial report [OPTIONS] [TASK]
```

Options

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

--stats
Display database statistics.

-d, --duration <duration>
Filter events for specified time period.

Options day|week|month|year|all

-s, --sort <sort>
Field to sort by.

Options task|time

-r, --reverse, --no-reverse
Reverse sort order.

--style <style>
Table output style.

Options simple|plain|grid|fancy_grid|github|pipe|orgtbl|jira|presto|psql|rst|mediawiki|moinmoin|youtrack|html|latex|latex_ra

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

RDIAL_SORT

Provide a default for *-s*

RDIAL_REVERSE

Provide a default for *-r*

run

Run command with timer.

```
rdial run [OPTIONS] [TASK]
```

Options

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

-n, --new
Start a new task.

-t, --time <time>
Set start time.

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

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

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

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

running

Display running task, if any.

```
rdial running [OPTIONS]
```

start

Start task.

```
rdial start [OPTIONS] [TASK]
```

Options

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

-c, --continue
Restart previous task.

-n, --new
Start a new task.

-t, --time <time>
Set start time.

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

stop

Stop task.

```
rdial stop [OPTIONS]
```

Options

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

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

--amend
Amend previous stop entry.

switch

Complete last task and start new one.

```
rdial switch [OPTIONS] [TASK]
```

Options

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

-n, --new
Start a new task.

-t, --time <time>
Set start time.

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

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

--amend
Amend previous stop entry.

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

timeclock

Generate ledger compatible timeclock file.

```
rdial timeclock [OPTIONS] [TASK]
```

Options

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

-d, --duration <duration>
Filter events for specified time period.

Options day|week|month|year|all

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

wrapper

Run predefined command with timer.

```
rdial wrapper [OPTIONS] [WRAPPER]
```

Options

-t, --time <time>
Set start time.

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

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

Arguments

WRAPPER
Optional argument

1.3 Getting started

1.3.1 Basic usage

The command interface is — I hope — 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.

1.3.2 Current task

The current task name is written to the database directory in the `.current` file. You could, for example, 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 *taskbar integration* document for some guidance on using **`rdial`** in various environments.

Manual section 1

Manual group user

1.4 rdial

1.4.1 Minimal time tracking for maximal benefit

1.4.2 SYNOPSIS

rdial [option]... <command>

1.4.3 DESCRIPTION

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.

1.4.4 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.

--colour, --no-colour
Output colourised informational text.

--help
Show help message and exit.

1.4.5 COMMANDS

rdial fsck

Check storage consistency.

```
rdial fsck [OPTIONS]
```

Options

-p, --progress, -q, --no-progress
Display progress bar.

rdial start

Start task.

```
rdial start [OPTIONS] [TASK]
```

Options

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

-c, --continue
Restart previous task.

-n, --new
Start a new task.

-t, --time <time>
Set start time.

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

rdial stop

Stop task.

```
rdial stop [OPTIONS]
```

Options

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

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

--amend
Amend previous stop entry.

rdial switch

Complete last task and start new one.

```
rdial switch [OPTIONS] [TASK]
```

Options

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

-n, --new
Start a new task.

-t, --time <time>
Set start time.

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

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

--amend
Amend previous stop entry.

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

rdial run

Run command with timer.

```
rdial run [OPTIONS] [TASK]
```

Options

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

-n, --new
Start a new task.

-t, --time <time>
Set start time.

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

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

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

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

rdial wrapper

Run predefined command with timer.

```
rdial wrapper [OPTIONS] [WRAPPER]
```

Options

-t, --time <time>
Set start time.

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

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

Arguments

WRAPPER
Optional argument

rdial report

Report time tracking data.

```
rdial report [OPTIONS] [TASK]
```

Options

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

--stats
Display database statistics.

-d, --duration <duration>
Filter events for specified time period.

Options day|week|month|year|all

-s, --sort <sort>
Field to sort by.

Options task|time

-r, --reverse, --no-reverse
Reverse sort order.

--style <style>
Table output style.

Options simple|plain|grid|fancy_grid|github|pipe|orgtbl|jira|presto|psql|rst|mediawiki|moinmoin|youtrack|html|latex|latex_ra

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

RDIAL_SORT

Provide a default for *-s*

RDIAL_REVERSE

Provide a default for *-r*

rdial running

Display running task, if any.

```
rdial running [OPTIONS]
```

rdial last

Display last event, if any.

```
rdial last [OPTIONS]
```

rdial ledger

Generate ledger compatible data file.

```
rdial ledger [OPTIONS] [TASK]
```

Options

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

-d, --duration <duration>
Filter events for specified time period.

Options day|week|month|year|all

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

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

RDIAL_RATE

Provide a default for *-r*

rdial timeclock

Generate ledger compatible timeclock file.

```
rdial timeclock [OPTIONS] [TASK]
```

Options

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

-d, --duration <duration>
Filter events for specified time period.

Options day|week|month|year|all

Arguments

TASK
Optional argument

Environment variables

RDIAL_TASK

Provide a default for *TASK*

1.4.6 BUGS

None known.

1.4.7 AUTHOR

Written by James Rowe

1.4.8 RESOURCES

Full documentation: https://rdial.readthedocs.io/

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

1.4.9 COPYING

1.5 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*.

1.5.1 File format

The configuration file is a `INI` format file. Use a section labelled `rdial` for global options, and a separate 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 `configparser`, and you can make use of all the features it provides(such as interpolation).

1.5.2 `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.

1.5.3 `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/' procrast
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 ever popular `procrast`-ination task.

1.6 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 `n`/`y` 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 `n`/`y` 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 `n`/`y` 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 `n`/`y` as its value.

RDIAL_PROFILE
: This controls whether to profile the execution of **rdial**. It must be a string value, and will be used as the profile's output filename.

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

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

RDIAL_SORT
: This controls the sorting order for output generated by **rdial report**. It can either `task` or `time` to sort by task name or cumulative time respectively.

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.

XDG_CONFIG_DIRS
: Stacked location of directories storing configuration files, see the XDG base directory specification for more information.

XDG_CONFIG_HOME
: Path to configuration files as given by the user, if unset use the default value of `$HOME/.config`. See the XDG base directory specification for more information.

1.7 Frequently Asked Questions

1.7.1 Why must I specify `--new` when creating a new task?

Perhaps you're super special and never produce a typo, I'm not. The *rdial start --new* option 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.

1.7.2 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 an exceedingly 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 useful backronym for it.

1.7.3 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 (Comma Separated Values) reader.

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

1.8 Integrating with taskbars

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

Note: You'll find exactly two examples right now, because these are the only two environments I use. Feel free to submit your own!

Todo: Add more fleshed out examples.

1.8.1 `awesomewm`

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

```
GLib = lgi.GLib
tasktext = wibox.widget.textbox!
tasktimer = with gears.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.

1.8.2 dwm

With dwm you're basically free to pump the status bar however you wish. If you're one of the users who likes to use a shell script to configure the bar, then you can just **cat** the `.current` file from within your script.

You could also edge towards mimicking the awesomewm configuration above with the following genie snippet leveraging glib:

```
[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 or rofi to bind to a key, the following zsh snippet shows how to build a selector for an existing task:

```
tasks=(${XDG_DATA_HOME:-~/.local/share}/rdial/*~*~(:t:s/.csv/))
rofi -dmenu -p "task?" <<< ${(F)tasks}
```

1.9 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 mainly as a reference for myself, and maybe to help out people who are already familiar with one of the entries below.

Todo: Check for recent additions to the arena.

1.9.1 arbtt

The "Automatic Rule-Based Time Tracker", where automatic could equally be awesome. You run the daemon and it records yours tasks automatically. The rules to configure switching are easy to write, and depending on your needs may well catch all your task switching.

It only falls down when you need to record tasks which aren't 100% about whacking away at the keyboard in a window, so if your time tracking needs are *entirely* screen based I recommend you try it out.

1.9.2 gtimelog

`gtimelog` is an interesting tool, and ticks many of the boxes for me. The ideas are quite well thought out, and the interface is 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 several of my co-workers stumbled across various bugs that dogged its usage. Unfortunately, the project is developed with bzr and launchpad, therefore it was simply abandoned in favour of trying to fix it. Most moved on to `org-mode`, and some to `rdial`.

1.9.3 hammertime

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

First, 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` if you're using a time based build tool like **`make`**. However, this could fixed 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 spent working on small projects.

1.9.4 hamster-time-tracker

ProjectHamster and its associated clients are a neat solution to time tracking. The gnome applet and command line interface are particularly polished. The **`hamster-cli`** in particular is extremely nice to use, and the built-in DWIM (Do What I Mean) date handling provides some great shortcuts for adding missing entries.

The data backend is a simple SQLite database, and is therefore very amenable to external processing. It provides tagging on top of the fields provided by **`rdial`**, and that alone may make it more useful to you.

1.9.5 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.

1.9.6 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!

1.9.7 taskwarrior

taskwarrior has some support for time tracking, 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.

1.9.8 timewarrior

From the same stable as taskwarrior, timewarrior is an *excellent* solution for time tracking. It doesn't support close messages, but it does have amazing tag support and some really useful query options.

There are import and export scripts in the `extra` subdirectory, and I truly recommend trying timewarrior out.

The only possible drawback is its speed, as with large numbers of records its startup becomes *very* slow. Exporting my eight year **rdial** database results in three second pauses on each **timew** run.

1.10 What is the future for rdial?

Where things stand NOW (2019-06-02):

```
$ time ./rdial.py --no-cache report --stats
65726 events in query
Duration of events 780 days, 14:04:22
First entry started at 2011-12-15 14:02:13
Last entry started at 2019-05-27 19:55:13.803240
Events exist on 2659 dates
Task "rdial" started 2019-05-27T19:55:13.803240Z
./rdial.py --no-cache report --stats  0.55s user 0.05s system 92% cpu 0.649 total
```

I'm clearly an extensive *rdial* user, and my inbox tells me that there are fair number of other — hopefully happy — users too. There are **no** plans to stop working on it, but there are some thoughts about where it could go.

1.10.1 Data storage

The output above is fairly representative of the speed of **rdial** *for me*[1]. All non-interactive commands run in well under three quarters of a second, and with the cache enabled the commands run within half a second. Performance *should* be better, but I find it to be within acceptable bounds.

One of the *original goals* of *rdial* was "easily accessible data, in a format that can be processed simply". The CSV-backed storage is easy to work with, and performance is acceptable. However, I'm more than a little tempted to replace it with a sqlite-backed solution because:

- sqlite is clearly a battle tested solution.
- Tests show performance improvements, although the large bottlenecks are in marshalling.
- The command line interface, **sqlite**, can export in various formats (including CSV).
- It would remove the need for most of the chicanery around loading and saving of data, probably significantly improving the stability.
- A small number of the reports that I generate with *rdial* are actually performed by importing the data in to a sqlite in-memory database via its CSV support, which proves the suitability already.

  Likelihood

  Effort

1.10.2 Language

I'm moderately happy with Python as the implementation language, although have considered switching a couple of times to improve command response times.

Over the years I've re-implemented *rdial* to various extents when playing with new languages:

Table 1: Possible implementation languages

Language	Pros	Cons
moonscript	• Nice language • Fast runtime • lua is available *everywhere*	• No significant uptake • Annoying split between lua versions, and of course luajit • Okay, not quite *everywhere* but close.
nim	• Fast runtime • Easy implementation	• Non-standard argument parsing, and no external package really fixes it • No significant uptake • Weak standard library, despite breadth
rust	• Fast runtime • Nice language • Good standard library • Common among co-worker users	• cargo, while **amazing**, doesn't fit my work environment very well • Doesn't *currently* work on all the arches I use

[1] If you have info or views on *your* **rdial** run times that you'd like to share I'd **love** to hear them.

Note: While clearly not new languages, both Genie and `C` have been considered too. `Genie` would be a good fit, but it is not clear what future it has. `C` would basically be either a *huge* amount of work or a simple wrapper around existing libraries[2], neither of which I find compelling.

I've also considered the git approach, with a *fast* top-level wrapper invoking external subcommands that could be in different languages. This way we can have *super* fast commands where they're really needed(**`rdial {start, stop, switch}`**), and choose a feature appropriate language for other commands(**`rdial report`** for instance). While attractive, it would make packaging *far* more annoying.

Likelihood

Effort

1.10.3 Interfaces

A couple of users have expressed an interest in alternative interfaces, but I'm not very enthusiastic about working on them myself. I understand the desire, but don't feel it is worth the effort. Added to that, if I'm not going to use them myself, I'm probably not the best person to produce them.

That said, I do use *`rdial`* outside of the command line, but entirely by wrapping the command line interface. This feels like a reasonable solution, but would benefit from a faster implementation and a *guaranteed* output format.

So, *my* intention would be:

- Adding full documentation for my awesomewm integration.
- Open sourcing the Android Wear swipe integrations for my watch.
- A faster implementation.
- *Guaranteed* stable output format.

Likelihood

Effort

1.11 Release HOWTO

1.11.1 Test

Tests can be run via pytest:

```
$ pip install -r extra/requirements-test.txt
$ pytest -v tests
```

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

1.11.2 Prepare release

With the tests passing, do the following steps:

- Update the version data in `rdial/_version.py`

[2] At which point it provides *no benefits* over `genie` for me.

- Update `NEWS.rst` with 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 a new release on GitHub

1.11.3 Update PyPI

Create and upload the new release tarballs to PyPI using twine:

```
$ ./setup.py sdist bdist_wheel
$ twine upload --sign dist/rdial-${version}*
```

Fetch the uploaded tarballs, and check for errors.

You should also test installation from PyPI, to check the experience *rdial*'s end users will have.

1.12 Todo items for documentation

Todo: Check for recent additions to the arena.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/rdial/checkouts/stable/doc/alternatives.rst, line 13.)

Todo: Add more fleshed out examples.

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

1.13 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.

1.13.1 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**(*_Event__task*, *start=None*, *delta=None*, *message=''*)

Initialise a new `Event` object.

Parameters

- **__task** – Task name to track
- **start** (Union[datetime, str, None]) – Start time for event
- **delta** (Union[timedelta, str, None]) – Duration for event
- **message** (Optional[str]) – Message to attach to event

running()
Check if event is running.

Return type Union[str, bool]

Returns Event name, if running

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

Parameters

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

Raises ***TaskNotRunningError*** – Event not running

Return type None

writer()
Prepare object for export.

Return type Dict[str, Optional[str]]

Returns Event data for object storage

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

Parameters

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

property dirty
Modified tasks requiring sync against storage.

Return type Iterable[str]

filter(*_Events__filt*)
Apply filter to events.

Parameters **__filt** – Function to filter with

Return type *Events*

Returns Events matching given filter function

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

Parameters

- **year** (int) – Year to filter on
- **month** (Optional[int]) – Month to filter on

- **day** (Optional[int]) – Day to filter on

Return type *Events*

Returns Events occurring within specified date

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

Parameters **__task** – Task name to filter on

Return type *Events*

Returns Events marked with given task name

for_week(*_Events__year*, *_Events__week*)
Filter events for a specific ISO (International Organization for Standardization)-8601 week.

Parameters

- **__year** – Year to filter events on
- **__week** – ISO-8601 week number to filter events on

Return type *Events*

Returns Events occurring in given ISO-8601 week

last()
Return current/last event.

Note: This handles the empty database case by returning None.

Return type Optional[*Event*]

Returns Last recorded event

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

Note: Assumes a new *Events* object should be created if the directory is missing.

Parameters

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

Return type *Events*

Returns Parsed events database

running()
Check if an event is running.

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

Return type Union[str, bool]

Returns Running event, if an event running

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

Parameters

- **__task** – Task name to track
- **new** (bool) – Whether to create a new task
- **start** (Union[datetime, str]) – ISO-8601 start time for event

Raises ***TaskRunningError*** – An event is already running

Return type None

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

Parameters

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

Raises ***TaskNotRunningError*** – No task running!

Return type None

sum()
Sum duration of all events.

Return type timedelta

Returns Sum of all event deltas

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

Return type List[str]

Returns Names of tasks in database

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

Parameters

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

Return type Iterator[*Events*]

write(*_Events__directory*)
Write database file.

Parameters **__directory** – Location to write database files to

Return type None

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'
```

1.13.2 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**(*ctx*, *globs*, *progress*)
Check storage consistency.

Parameters

- **ctx** – Current command context
- **globs** – Global options object
- **progress** – Display progressbar

rdial.cmdline.**start**(*globs*, *task*, *continue*, *new*, *time*)
Start task.

Parameters

- **globs** – Global options object
- **task** – Task name to operate on
- **continue_** – Pull task name from last running task
- **new** – Create a new task

- **time** – Task start time

rdial.cmdline.**stop**(*globs*, *message*, *fname*, *amend*)
Stop task.

Parameters

- **globs** – Global options object
- **message** – Message to assign to event
- **fname** – Filename to read message from
- **amend** – Amend a previously stopped event

rdial.cmdline.**switch**(*globs*, *task*, *new*, *time*, *amend*, *message*, *fname*)
Complete last task and start new one.

Parameters

- **globs** – Global options object
- **task** – Task name to operate on
- **new** – Create a new task
- **time** – Task start time
- **amend** – Amend a previously stopped event
- **message** – Message to assign to event
- **fname** – Filename to read message from

rdial.cmdline.**run**(*globs*, *task*, *new*, *time*, *message*, *fname*, *command*)
Run command with timer.

Parameters

- **globs** – Global options object
- **task** – Task name to operate on
- **new** – Create a new task
- **time** – Task start time
- **message** – Message to assign to event
- **fname** – Filename to read message from
- **command** – Command to run

rdial.cmdline.**wrapper**(*ctx*, *globs*, *time*, *message*, *fname*, *wrapper*)
Run predefined command with timer.

Parameters

- **ctx** – Click context object
- **globs** – Global options object
- **time** – Task start time
- **message** – Message to assign to event
- **fname** – Filename to read message from
- **wrapper** – Run wrapper to execute

rdial.cmdline.**report**(*globs*, *task*, *stats*, *duration*, *sort*, *reverse*, *style*)
Report time tracking data.

Parameters

- **globs** – Global options object
- **task** – Task name to operate on
- **stats** – Display short overview of data
- **duration** – Time window to filter on
- **sort** – Key to sort events on
- **reverse** – Reverse sort order
- **style** – Table formatting style

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

Parameters **globs** – Global options object

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

Parameters **globs** – Global options object

rdial.cmdline.**ledger**(*globs*, *task*, *duration*, *rate*)
Generate ledger compatible data file.

Parameters

- **globs** – Global options object
- **task** – Task name to operate on
- **duration** – Time window to filter on
- **rate** – Rate to assign hours in report

rdial.cmdline.**timeclock**(*globs*, *task*, *duration*)
Generate ledger compatible timeclock file.

Parameters

- **globs** – Global options object
- **task** – Task name to operate on
- **duration** – Time window to filter on

Entry points

rdial.cmdline.**cli**(*ctx*, *directory*, *backup*, *cache*, *config*, *interactive*, *colour*)
Main command entry point.

Parameters

- **ctx** – Current command context
- **directory** – Location to store event data
- **backup** – Whether to create backup files
- **cache** – Whether to create cache files

- **config** – Location of config file
- **interactive** – Whether to support interactive message editing
- **colour** – Whether to colourise output

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

Return type int

Returns Final exit code

Command support

rdial.cmdline.**filter_events**(*__globs*, *__task=None*, *__duration='all'*)
Filter events for report processing.

Parameters

- **__globs** (ROAttrDict) – Global options object
- **__task** (Optional[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** (*Event*) – Current task
- **__edit** (bool) – Whether to edit existing message

Return type str

Returns Message to use

CLI support

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

convert(*_TaskNameParamType__value*, *_TaskNameParamType__param*, *_TaskNameParamType__ctx*)
Check given task name is valid.

Parameters

- **__value** – Value given to flag
- **__param** – Parameter being processed
- **__ctx** – Current command context

Return type str

Returns Valid task name

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

convert(*_StartTimeParamType__value*, *_StartTimeParamType__param*, *_StartTimeParamType__ctx*)
Check given start time is valid.

Parameters

- **__value** – Value given to flag
- **__param** – Parameter being processed
- **__ctx** – Current command context

Return type datetime

Returns Valid start time

rdial.cmdline.**task_from_dir**(*__ctx*, *__param*, *__value*)
Override task name default using name of current directory.

Parameters

- **__ctx** (Context) – Current command context
- **__param** (Option) – Parameter being processed
- **__value** (bool) – True if flag given

Return type None

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

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

Parameters **__fun** (Callable) – Function to add options to

Return type Callable

Returns Function with additional options

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

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

Parameters **__fun** (Callable) – Function to add options to

Return type Callable

Returns Function with additional options

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

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

Parameters **__fun** (Callable) – Function to add options to

Return type Callable

Returns Function with additional options

1.13.3 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.**read_config**(*user_config=None*, *cli_options=None*)
Read configuration data.

Parameters

- **user_config** (Optional[str]) – User defined config file
- **cli_options** (Optional[Dict[str, Union[bool, str]]]) – Command line options

Return type ConfigParser

Returns Parsed configuration data

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

See also:

Integrating with taskbars

Parameters **__fun** (Callable) – Function to wrap

Return type Callable

Returns Wrapped function

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

See also:

Integrating with taskbars

Parameters **__fun** (Callable) – Function to wrap

Return type Callable

Returns Wrapped function

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

Return type bool

Returns True if __fname is newer than __reference

rdial.utils.**term_link**(*__target*, *name=None*)
Generate a terminal hyperlink.

See https://gist.github.com/egmontkob/eb114294efbcd5adb1944c9f3cb5feda.

Parameters

- **__target** (str) – Hyperlink target
- **name** (Optional[str]) – Target name

Return type str

Returns Formatted hyperlink for terminal output

Time handling

rdial.utils.**iso_week_to_date**(*__year*, *__week*)
Generate date range for a given ISO-8601 week.

ISO-8601 defines a week as Monday to Sunday, with the first week of a year being the first week containing a Thursday.

Parameters

- **__year** (int) – Year to process
- **__week** (int) – Week number to process

Return type Tuple[date, date]

Returns Date range objects for given week

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

Return type datetime

Returns Parsed datetime object

Development tools

rdial.utils.**maybe_profile**()
Profile the wrapped code block.

When *RDIAL_PROFILE* is set execute the enclosed block under bprofile. The envvar's value should be the name of the output file to generate.

When *RDIAL_PROFILE* is unset, this is just a no-op.

Return type Abstractcontextmanager[+T_co]

Examples

Time handling

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

Development tools

```
>>> with maybe_profile():
...     time.sleep(10)
```

1.13.4 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.**TaskNotExistError**
Exception for attempting to operate on a non-existing task.

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!
```

1.14 Glossary

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

B

C

D

E

F

G

I

L

M

N

P

R

S

T

W

X