Todoman

Release 4.2.1

Hugo Osvaldo Barrera <hugo@barrera.io>, et al

Mar 25, 2023

CONTENTS

Todoman is a simple, standards-based, cli todo (aka: task) manager. Todos are stored into icalendar files, which means you can sync them via CalDAV using, for example, vdirsyncer.

Todoman is now part of the `pimutils` project, and is hosted at GitHub.

CHAPTER ONE

FEATURES

- Listing, editing and creating todos.
- Todos are read from individual ics files from the configured directory. This matches the vdir specification.
- There's support for the most common TODO features for now (summary, description, location, due date and priority).
- Runs on any Unix-like OS. It's been tested on GNU/Linux, BSD and macOS.
- Unsupported fields may not be shown but are *never* deleted or altered.

CHAPTER

TWO

CONTRIBUTING

See *Contributing* for details on contributing.

CHAPTER THREE

CAVEATS

Support for the `percent-completed` attribute is incomplete. Todoman can only mark todos as completed (100%), and will not reflect nor allow editing for values for `percent > 0 ^ percent < 100`.

CHAPTER FOUR

TABLE OF CONTENTS

4.1 Installing

4.1.1 Distribution packages

If todoman is packaged for your OS/distribution, using your system's standard package manager is probably the easiest way to install todoman.

ArchLinux

todoman is packaged in the community repository, and can be installed using:

```
pacman -S todoman
```

macOS

todoman is packaged in homebrew, and can be installed using:

```
brew install todoman
```

4.1.2 Installation via PIP

Since *todoman* is written in python, you can use python's package managers, *pip* by executing:

```
pip install todoman
```

or the latest development version by executing:

```
pip install git+git://github.com/pimutils/todoman.git
```

This should also take care of installing all required dependencies.

4.1.3 Manual installation

If pip is not available either (this is most unlikely), you'll need to download the source tarball and install via setup.py, though this is not a recommended installation method:

```
python3 setup.py install
```

4.1.4 bash autocompletion (optional)

There is an autocompletion function for bash provided in the `contrib` directory. If you want to enable autocompletion for todoman in bash, copy the file `contrib/autocompletion/bash/_todo` to any directory you want. Typically `/etc/bash_completion.d` is used for system-wide installations or `~/.bash_completion.d` for local installations. In the former case, the file is automatically sourced in most distributions, in the latter case, you will most likely need to add:

```
source ~/.bash_completion.d/_todo
```

to your `~/.bashrc`.

4.1.5 zsh autocompletion (optional)

There is an autocompletion function for zsh provided in the `contrib` directory. If you want to enable autocompletion for todoman in zsh, copy the file `contrib/autocompletion/zsh/_todo` to any directory in your `$fpath`. Typically `/usr/local/share/zsh/site-functions/` is used for system-wide installations.

4.1.6 Requirements

Todoman requires python 3.8 or later. Installation of required libraries can be done via pip, or your OS's package manager.

Recent versions also have experimental support for pypy3.

4.1.7 Notes for Packagers

All of todoman's dependencies are listed in the `install_requires` section of the setup.py file. New dependencies will be clearly announced in the `CHANGELOG.rst` file for each release. Patch releases (eg: those where only the third digit of the version is incremented) **will not** introduce new dependencies.

If your packages are generated by running `setup.py install` or some similar mechanism, you'll end up with a very slow entry point (that's the `/usr/bin/todo` file). Package managers should use the file included in this repository under `bin/todo` and replace the above one.

The root cause of the issue is really how python's setuptools generates these and outside of the scope of this project.

If your packages are generated using python wheels, this should not be an issue (much like it won't be an issue for users installing via `pip`).

Additionally, *jq* is dependency for zsh's autocompletion. For platforms where *zsh* is the default shell, it is recommended to list *jq* as a dependency, for others adding it as an optional dependency should suffice.

4.2 Configuring

You’ll need to configure Todoman before the first usage, using its simple ini-like configuration file.

4.2.1 Configuration File

The configuration file should be placed in `$XDG_CONFIG_HOME/todoman/config.py`. `$XDG_CONFIG_HOME` defaults to `~/.config` is most situations, so this will generally be `~/.config/todoman/config.py`.

`cache_path`

The location of the cache file (an sqlite database). This file is used to store todo data and speed up execution/startup, and also contains the IDs for todos. If the value is not specified, a path relative to `$XDG_CACHE_HOME` will be used. `$XDG_CACHE_HOME` generally resolves to `~/.cache/`.

type
str

default
`"$XDG_CACHE_HOME/todoman/cache.sqlite3"`

`color`

By default todoman will disable colored output if stdout is not a TTY (value `auto`). Set to `never` to disable colored output entirely, or `always` to enable it regardless. This can be overridden with the `--color` option.

type
str

default
`"auto"`

`date_format`

The date format used both for displaying dates, and parsing input dates. If this option is not specified the system locale’s is used.

type
str

default
`"%x"`

`default_command`

When running `todo` with no commands, run this command.

type
str

default
`"list"`

`default_due`

The default difference (in hours) between new todo’s due date and creation date. If not specified, the value is 24. If set to 0, the due date for new todos will not be set.

type
int

default
`24`

`default_list`

The default list for adding a todo. If you do not specify this option, you must use the `--list` / `-l` option every time you add a todo.

type
str

default
`None`

`default_priority`

The default priority of a task on creation. Highest priority is 1, lowest priority is 10, and 0 means no priority at all.

type
int

default
`None`

`dt_separator`

The string used to separate date and time when displaying and parsing.

type
str

default
`" "`

`humanize`

If set to true, datetimes will be printed in human friendly formats like "tomorrow", "in one hour", "3 weeks ago", etc.

If false, datetimes will be formatted using `date_format` and `time_format`.

type
bool

default
`False`

`path`

A glob pattern matching the directories where your todos are located. This pattern will be expanded, and each matching directory (with any icalendar files) will be treated as a list.

type
str

default
None, this field is mandatory.

`startable`

If set to true, only show todos which are currently startable; these are todos which have a start date today, or some day in the past. Todos with no start date are always considered current. Incomplete todos (eg: partially-complete) are also included.

type
bool

default
`False`

time_format

The date format used both for displaying times, and parsing input times.

type
str

default
"%X"

4.2.2 Sample configuration

The below example should serve as a reference. It will read ics files from any directory inside ~/.local/share/calendars/, uses the ISO-8601 date format, and set the due date for new todos in 48hs.

```
# A glob expression which matches all directories relevant.
path = "~/.local/share/calendars/*"
date_format = "%Y-%m-%d"
time_format = "%H:%M"
default_list = "Personal"
default_due = 48
```

4.2.3 Color and displayname

- You can set a color for each task list by creating a `color` file containing a color code in the hex format: `#RRGGBB`.
- A file named `displayname` indicates how the task list should be named and is needed when there are multiple directories sharing a name, e.g.: when using multiple $CloudInstances. The default is the directory name.

See also relevant documentation for the vdir format.

4.2.4 Timezone

Todoman will use the system-wide configured timezone. If this doesn't work for you, you _may_ override the timezone by specifying the `TZ` environment variable.

For instruction on changing your system's timezone, consult your distribution's documentation.

4.3 Usage

Todoman usage is CLI based (thought there are some TUI bits, and the intentions is to also provide a fully TUI-based interface).

The default action is `list`, which outputs all tasks for all calendars, each with a semi-permanent unique id:

```
1 [ ] !!! 2015-04-30 Close bank account @work (0%)
2 [ ] !              Send minipimer back for warranty replacement @home (0%)
3 [X]     2015-03-29 Buy soy milk @home (100%)
4 [ ] !!             Fix the iPad's screen @home (0%)
5 [ ] !!             Fix the Touchpad battery @work (0%)
```

The columns, in order, are:

- An id.

- Whether the task has been completed or not.
- An `!!!` indicating high priority, `!!` indicating medium priority, `!` indicating low priority tasks.
- The due date.
- The task summary.
- The list the todo is from; it will be hidden when filtering by one list, or if the database only contains a single list.
- The completed percentage.

The id is retained by `todoman` until the next time you run the `flush` command.

To operate on a todo, the id is what's used to reference it. For example, to edit the *Buy soy milk* task from the example above, the proper command is `todo edit 3`, or `todo undo 3` to un-mark the task as done.

Editing tasks can only be done via the TUI interface for now, and cannot be done via the command line yet.

4.3.1 Synchronization

If you want to synchronize your tasks, you'll need something that syncs via CalDAV. vdirsyncer is the recommended tool for this.

4.3.2 Interactive shell

If you install click-repl, todoman gets a new command called `repl`, which launches an interactive shell with tab-completion.

4.3.3 Integrations

When attempting to integrate `todoman` into other systems or parse its output, you're advised to use the `--porcelain` flag, which will print all output in a pre-defined format that will remain stable regardless of user configuration or version.

The format is JSON, with a single array containing each todo as a single entry (object). Fields will always be present; if a todo does not have a value for a given field, it will be printed as `null`.

Fields MAY be added in future, but will never be removed.

Conky

Todoman can be used with Conky by using one of the shell execution variables. Given the nature of pimutils utilities, there is no need to query new information every time conky updates so `execi` will be the best option most of the time.

Adding `${execi 30 todo}` inside the text section will display the output of the command and update it every 30 seconds.

A working configuration can be found here.

4.3.4 Sorting

The tasks can be sorted with the `--sort` argument. Sorting may be done according to the following fields:

- `description`
- `location`
- `status`
- `summary`
- `uid`
- `rrule`
- `percent_complete`
- `priority`
- `sequence`
- `categories`
- `completed_at`
- `created_at`
- `dtstamp`
- `start`
- `due`
- `last_modified`

4.4 Man page

4.4.1 Description

Todoman is a simple, standards-based, cli todo (aka: task) manager. Todos are stored into *icalendar* files, which means you can sync them via *CalDAV* using, for example, *vdirsyncer*.

4.4.2 Usage

4.5 Usage

Todoman usage is CLI based (thought there are some TUI bits, and the intentions is to also provide a fully TUI-based interface).

The default action is `list`, which outputs all tasks for all calendars, each with a semi-permanent unique id:

```
1 [ ] !!! 2015-04-30 Close bank account @work (0%)
2 [ ] !              Send minipimer back for warranty replacement @home (0%)
3 [X]     2015-03-29 Buy soy milk @home (100%)
4 [ ] !!             Fix the iPad's screen @home (0%)
5 [ ] !!             Fix the Touchpad battery @work (0%)
```

The columns, in order, are:

- An id.
- Whether the task has been completed or not.
- An `!!!` indicating high priority, `!!` indicating medium priority, `!` indicating low priority tasks.
- The due date.
- The task summary.
- The list the todo is from; it will be hidden when filtering by one list, or if the database only contains a single list.
- The completed percentage.

The id is retained by `todoman` until the next time you run the `flush` command.

To operate on a todo, the id is what's used to reference it. For example, to edit the *Buy soy milk* task from the example above, the proper command is `todo edit 3`, or `todo undo 3` to un-mark the task as done.

Editing tasks can only be done via the TUI interface for now, and cannot be done via the command line yet.

4.5.1 Synchronization

If you want to synchronize your tasks, you'll need something that syncs via CalDAV. vdirsyncer is the recommended tool for this.

4.5.2 Interactive shell

If you install click-repl, todoman gets a new command called `repl`, which launches an interactive shell with tab-completion.

4.5.3 Integrations

When attempting to integrate `todoman` into other systems or parse its output, you're advised to use the `--porcelain` flag, which will print all output in a pre-defined format that will remain stable regardless of user configuration or version.

The format is JSON, with a single array containing each todo as a single entry (object). Fields will always be present; if a todo does not have a value for a given field, it will be printed as `null`.

Fields MAY be added in future, but will never be removed.

Conky

Todoman can be used with Conky by using one of the shell execution variables. Given the nature of pimutils utilities, there is no need to query new information every time conky updates so `execi` will be the best option most of the time.

Adding `${execi 30 todo}` inside the text section will display the output of the command and update it every 30 seconds.

A working configuration can be found here.

4.5.4 Sorting

The tasks can be sorted with the `--sort` argument. Sorting may be done according to the following fields:

- `description`
- `location`
- `status`
- `summary`
- `uid`
- `rrule`
- `percent_complete`
- `priority`
- `sequence`
- `categories`
- `completed_at`
- `created_at`
- `dtstamp`
- `start`
- `due`
- `last_modified`

4.5.5 Configuring

4.6 Configuring

You'll need to configure Todoman before the first usage, using its simple ini-like configuration file.

4.6.1 Configuration File

The configuration file should be placed in `$XDG_CONFIG_HOME/todoman/config.py`. `$XDG_CONFIG_HOME` defaults to `~/.config` is most situations, so this will generally be `~/.config/todoman/config.py`.

`cache_path`

The location of the cache file (an sqlite database). This file is used to store todo data and speed up execution/startup, and also contains the IDs for todos. If the value is not specified, a path relative to `$XDG_CACHE_HOME` will be used. `$XDG_CACHE_HOME` generally resolves to `~/.cache/`.

type
str

default
`"$XDG_CACHE_HOME/todoman/cache.sqlite3"`

`color`

By default todoman will disable colored output if stdout is not a TTY (value `auto`). Set to `never` to disable colored output entirely, or `always` to enable it regardless. This can be overridden with the `--color` option.

type
str

default
`"auto"`

`date_format`

The date format used both for displaying dates, and parsing input dates. If this option is not specified the system locale's is used.

type
str

default
`"%x"`

`default_command`

When running `todo` with no commands, run this command.

type
str

default
`"list"`

`default_due`

The default difference (in hours) between new todo's due date and creation date. If not specified, the value is 24. If set to 0, the due date for new todos will not be set.

type
int

default
`24`

`default_list`

The default list for adding a todo. If you do not specify this option, you must use the `--list` / `-l` option every time you add a todo.

type
str

default
`None`

`default_priority`

The default priority of a task on creation. Highest priority is 1, lowest priority is 10, and 0 means no priority at all.

type
int

default
`None`

`dt_separator`

The string used to separate date and time when displaying and parsing.

type
str

default
`" "`

`humanize`

If set to true, datetimes will be printed in human friendly formats like "tomorrow", "in one hour", "3 weeks ago", etc.

If false, datetimes will be formatted using `date_format` and `time_format`.

type
bool

default
`False`

`path`

A glob pattern matching the directories where your todos are located. This pattern will be expanded, and each matching directory (with any icalendar files) will be treated as a list.

type
str

default
None, this field is mandatory.

`startable`

If set to true, only show todos which are currently startable; these are todos which have a start date today, or some day in the past. Todos with no start date are always considered current. Incomplete todos (eg: partially-complete) are also included.

type
bool

default
`False`

`time_format`

The date format used both for displaying times, and parsing input times.

type
str

default
`"%X"`

4.6.2 Sample configuration

The below example should serve as a reference. It will read ics files from any directory inside `~/.local/share/calendars/`, uses the ISO-8601 date format, and set the due date for new todos in 48hs.

```
# A glob expression which matches all directories relevant.
path = "~/.local/share/calendars/*"
date_format = "%Y-%m-%d"
time_format = "%H:%M"
default_list = "Personal"
default_due = 48
```

4.6.3 Color and displayname

- You can set a color for each task list by creating a `color` file containing a color code in the hex format: `#RRGGBB`.
- A file named `displayname` indicates how the task list should be named and is needed when there are multiple directories sharing a name, e.g.: when using multiple $CloudInstances. The default is the directory name.

See also relevant documentation for the vdir format.

4.6.4 Timezone

Todoman will use the system-wide configured timezone. If this doesn't work for you, you _may_ override the timezone by specifying the `TZ` environment variable.

For instruction on changing your system's timezone, consult your distribution's documentation.

4.6.5 Caveats

Support for the `percent-completed` attribute is incomplete. Todoman can only mark todos as completed (100%), and will not reflect nor allow editing for values for `percent > 0 ^ percent < 100`.

4.6.6 Contributing

For information on contributing, see: https://todoman.readthedocs.io/en/stable/contributing.html

4.6.7 LICENCE

Todoman is licensed under the ISC licence. See LICENCE for details.

4.7 Contributing

Bug reports and code and documentation patches are greatly appreciated. You can also help by using the development version of todoman and reporting any bugs you might encounter here.

All participants must follow the pimutils Code of Conduct.

Before working on a new feature or a bug, please browse existing issues to see whether it has been previously discussed. If the change in question is a bigger one, it's always good to open a new issue to discuss it before your starting working on it.

4.7.1 Hacking

Runtime dependencies are listed in `setup.py`. We recommend that you use virtualenv to make sure that no additional dependencies are required without them being properly documented. Run `pip install -e .` to install todoman and its dependencies into a virtualenv.

We use `pre-commit` to run style and convention checks. Run `pre-commit install`` to install our git-hooks. These will check code style and inform you of any issues when attempting to commit. This will also run `black` to reformat code that may have any inconsistencies.

Commits should follow Git Commit Guidelines whenever possible, including rewriting branch histories to remove any noise, and using a imperative present tense for commit summary messages (50 characters maximum).

All commits should pass all tests to facilitate bisecting in future.

An overview of the Todo lifecycle

This is a brief overview of the life cycles of todos (from the apps point of view) as they are read from disk, displayed, and or saved again.

When the app starts, it will read all todos from disk, and initialize from the cache any further display (either `list`, `show`, `edit`, etc) is then done reading from the cache, which only contains the fields with which we operate. This stage also assigns the id numbers to each todo.

When a Todo is edited, the entire cycle is:

- File is read from disk and cached (if not already cached).
- A Todo object is created by reading the cache.
- If edition is interactive, show the UI now.
- No matter how the edition occurs, apply changes to the Todo object.
- **Start saving process:**
    - Read file from disk (as a VTodo object).
    - Apply changes from fields to the VTodo object.
    - Write to disk.

The main goal of this is to simplify how many conversions we have. If we read from disk to the editor, we'd need an extra VTodo->Todo conversion code that skips the cache.

Running and testing locally

The easiest way to run tests, it to install `tox`, and then simply run `tox`. By default, several python versions and environments are tested. If you want to run a specific one use `tox -e ENV`, where `ENV` should be one of the environments listed by `tox -l`.

See the tox documentation for further details.

To run your modified copy of `todoman` without installing it, it's recommended you set up a virtualenv, and run `pip install -e .` to install your checked-out copy into it (this'll make `todo` run your local copy while the virtualenv is active).

Authorship

Authors may add themselves to `AUTHORS.rst`, and all copyright is retained by them. Contributions are accepted under the *ISC licence*.

4.7.2 Patch review checklist

Please follow this checklist when submitting new PRs (or reviewing PRs by others):

CI will automatically check these for us:

1. Do all tests pass?
2. Does the documentation build?
3. Do all linting and style checks pass?

Please keep an eye open for these other items:

1. If any features have changed, make sure the docs reflect this.
2. If there are any user-facing changes, make sure the *Changelog* reflects this.
3. If there are any dependency changes, make sure the *Changelog* reflects this.
4. If not already present, please add yourself to `AUTHORS.rst`.

4.7.3 Packaging

We appreciate distributions packaging todoman. Packaging should be relatively straightforward following usual Python package guidelines. We recommend that you git-clone tags, and build from source, since these tags are GPG signed.

Dependencies are listed in `setup.py`. Please also try to include the extras dependencies as optional dependencies (or what maps best for your distribution).

You'll need to run `python setup.py build` to generate the `todoman/version.py` file which is necessary at run-time.

We recommend that you include the *Man page* in distribution packages. You can build this by running:

```
sphinx-build -b man docs/source docs/build/man
```

The man page will be saved as *docs/build/man/todo.1*.

Generating the man pages requires that todoman and its doc dependencies (see `requirements-docs.txt`) are either installed, or in the current `PYTHONPATH`.

4.8 Changelog

This file contains a brief summary of new features and dependency changes or releases, in reverse chronological order.

4.8.1 v4.2.0

- Added full support for categories.

4.8.2 v4.1.0

- The "table" layout has been dropped in favour of a simpler, fluid layout. As such, `tabulate` is not longer a required dependency.
- Added support for python 3.10.

4.8.3 v4.0.1

- Fix issue passing arguments to `default_command`.
- Various documentation improvements.

4.8.4 v4.0.0

Breaking changes in the configuration format

The configuration format is changing with release 4.0.0. We currently depend on an unmaintained library for configuration. It's not currently in a working state, and while some distributions are patching it, setting up a clean environment is a bit non-trivial, and the situation will only degrade in future.

The changes in format are be subtle, and also come with an intention to add further extensibility in future. Configuration files will be plain python. If you don't know Python don't worry, you don't _need_ to know Python.

I'll take my own config as a reference. The pre-4.0.0 format is:

`` `dosini [main] path = ~/.local/share/calendars/* time_format = '%H:%M' default_list = todo humanize = true startable = true ` ``

The 4.0.0 version would look like this:

`` `python path = "~/.local/share/calendars/*" time_format = "%H:%M" default_list = "todo" humanize = True startable = True ` ``

Key differences:

- The *[main]* header is no longer needed.
- All strings must be quoted (this was previously optional).
- True and False start with uppercase.
- Using *yes* or *on* is no longer valid; only *True* and *False* are valid.

That's basically it. This lets up drop the problematic dependency, and we don't actually need anything to read the config: it's just python code like the rest of *todoman*!

For those users who _are_ python developers, you'll note this gives some interesting flexibility: you CAN add any custom python code into the config file. For example, you can defined the *path* programatically:

```python
def get_path() -> str:
    ...

path = get_path
```

Dropped support

- Dropped support older Python versions. Only 3.8 and 3.9 are now supported.

Minor changes

- Added support for python 3.9.
- The dependency *configobj* is no longer required.
- Click 8.0 is now supported.
- Fix crash when `default_command` has arguments.

4.8.5 v3.9.0

- The man page has been improved. `sphinx-click` is now required to build the documentation.

4.8.6 v3.8.0

- Don't display list if there's only one list (of one list has been specified).
- Fixed several issues when using dates with no times.
- Dropped support for Python 3.4.

4.8.7 v3.7.0

- Properly support iCal files with dates (instead of datetimes).

4.8.8 v3.6.0

- Allow passing a custom configuration file with the `--config/-c` option.
- Cached list metadata is now invalidated when it has changed on-disk.
- Support for click < 6.0 has been dropped (it wasn't actually working perfectly any more anyway). Click 7.x is the only currently supported version.
- `click-repl` is now listed as an optional dependency. It is required for the `todo repl` command.
- Add the `default_priority` config setting.
- Drop support for Python 3.4.

4.8.9 v3.5.0

- Fix crashes due to API changes in icalendar 4.0.3.
- Dropped compatibility for icalendar < 4.0.3.

4.8.10 v3.4.1

- Support Python 3.7.
- Support click>=7.0.
- Properly parse RBGA colours.

4.8.11 v3.4.0

- Add `-r` option to `new` to read description from `stdin`.
- Add a dedicated zsh completion function.
- Lists matching is now case insensitive, unless there are multiple lists with colliding names, in which case those will be treated case-sensitive.
- Fix some tests that failed due to test dependency changes.

4.8.12 v3.3.0

- New runtime dependency: `click-log`.
- Drop support for Python 3.3, which has reached its end of life cycle.
- Add *–raw* flag to *edit*. This allows editing the raw icalendar file, but **only use this if you really know what you're doing**. There's a big risk of data loss, and this is considered a developer / expert feature!

4.8.13 v3.2.4

- Deploy new versions to PyPI using `twine`. Travis doesn't seem to be working.

4.8.14 v3.2.3

- Tests should no longer fail with `pyicu` installed.
- Improved documentation regarding how to test locally.

4.8.15 v3.2.2

- Initial support for (bash) autocompletion.
- The location field is not printed as part of `--porcelain`.

4.8.16 v3.2.1

- Fix start-up crash caused by click_log interface change.
- Dropped runtime dependency: `click_log`.

4.8.17 v3.2.0

- Completing recurring todos now works as expected and does not make if disappear forever.

4.8.18 v3.1.0

- Last-modified fields of todos are now updated upon edition.
- Sequence numbers are now properly increased upon edition.
- Add new command `todo cancel` to cancel an existing todo without deleting it.
- Add a new setting `default_command`.
- Replace `--all` and `--done-only` with `--status`, which allows fine-grained status filtering. Use `--status ANY` or `--status COMPLETED` to obtain the same results as the previous flags.
- Rename `--today` flag to `--startable`.
- Illegal start dates (eg: start dates that are not before the due date) are ignored and are removed when saving an edited todo.

4.8.19 v3.0.1

- Fix a crash for users upgrading from pre-v3.0.0, caused due to the cache's schema not being updated.

4.8.20 v3.0.0

New features

- Add a `today` setting and flag to exclude todos that start in the future.
- Add the `--humanize` to show friendlier date times (eg: `in 3 hours`).
- Drop `--urgent` and introduced `--priority`, which allows fine-filtering by priority.
- Add support for times in due dates, new `time_format` setting.
- Use the system's date format as a default.
- Add list selector to the interactive editor.
- Add `--start=[before|after] [DATE]` option for `list` to only show todos starting before/after given date.
- Add flag "–done-only" to todo list. Displays only completed tasks.

- Make the output of move, delete, copy and flush consistent.
- Porcelain now outputs proper JSON, rather than one-JSON-per-line.
- Increment sequence number upon edits.
- Print a descriptive message when no lists are found.
- Add full support for locations.

Packaging changes

- New runtime dependency: `tabulate`.
- New runtime dependency: `humanize`.
- New supported python version: `pypy3`.
- Include an alternative [much faster] entry point (aka "bin") which we recommend all downstream packagers use. Please see the *Notes for Packagers* documentation for further details.

4.8.21 v2.1.0

- The global `--verbosity` option has been introduced. It doesn't do much for now though, because we do not have many debug logging statements.
- New PyPI dependency `click-log`.
- The `--no-human-time` flag is gone. Integrations/scripts might want to look at `--porcelain` as an alternative.
- Fix crash when running `todo new`.
- Fixes some issues when filtering todos from different timezones.
- Attempt to create the cache file's directory if it does not exist.
- Fix crash when running `--porcelain show`.
- Show `id` for todos everywhere (eg: including new, etc).
- Add the `ctrl-s` shortcut for saving in the interactive editor.

4.8.22 v2.0.2

- Fix a crash after editing or completing a todo.

4.8.23 v2.0.1

- Fix a packaging error.

4.8.24 v2.0.0

- New flag `--porcelain` for programmatic integrations to use. See the `integrations` section *here* for details.
- Implement a new *configuration option*: `default_due`.
- The configuration file is now pre-emptively validated. Users will be warned of any inconsistencies.
- The `list` command has a new `--due` flag to filter tasks due soon.
- Todo ids are now persisted in a cache. They can be manually purged using `flush`.
- New runtime dependency: configobj
- New runtime dependency: python-dateutil
- New test dependency: flake8-import-order.

4.9 Licence

Todoman is licensed under the ISC licence:

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search