todo.py Documentation

Release development

Ian Cordasco

January 17, 2016

Contents

`todo.py` is a python script which aims to be an active and accurate alternative to Gina Trapani's `todo.sh`. `todo.py` is extensible with *add-ons* that can be written in a few ways, and written in a way that should be inviting to hackers of varying degrees of experience.

CHAPTER 1

Usage

```
Usage: todo.py [options] action [arg(s)]

Options:
  -h, --help            show this help message and exit
  -c CONFIG, --config=CONFIG
                        Supply your own configuration file,must be an absolute path
  -d TODO_DIR, --dir=TODO_DIR
                        Directory you wish ./todo.py to use.
  -p, --plain-mode      Toggle coloring of items
  -P, --no-priority     Toggle display of priority labels
  -t, --prepend-date    Toggle whether the date is prepended to new items.
  -V, --version         Print version, license, and credits
  -i, --invert-colors   Toggle coloring the text of items or background of items.
  -l, --legacy          Toggle organization of items in the old manner.
  -+                    Toggle display of +projects in-line with items.
  -@                    Toggle display of @contexts in-line with items.
  -#                    Toggle display of #{dates} in-line with items.
```

```
Use ./todo.py -h for option help

Usage: ./todo.py command [arg(s)]
   add | a "Item to do +project @context #{yyyy-mm-dd}"
      Adds 'Item to do +project @context #{yyyy-mm-dd}' to your todo.txt
      file.
      +project, @context, #{yyyy-mm-dd} are optional

   addm "First item to do +project @context #{yyyy-mm-dd}
      Second item to do +project @context #{yyyy-mm-dd}
      ...
      Last item to do +project @context #{yyyy-mm-dd}
      Adds each line as a separate item to your todo.txt file.

   append | app NUMBER "text to append"
      Append "text to append" to item NUMBER.

   del | rm NUMBER
      Deletes the item on line NUMBER in todo.txt

   depri | dp NUMBER
      Remove the priority of the item on line NUMBER.
   do NUMBER
      Marks item with corresponding number as done and moves it to
      your done.txt file.

   help | h
      Display this message and exit.

   list | ls
      Lists all items in your todo.txt file sorted by priority.

   listcon | lsc
      Lists all items in your todo.txt file sorted by context.

   listdate | lsd
      Lists all items in your todo.txt file sorted by date.

   listproj | lsp
      Lists all items in your todo.txt file sorted by project title.

   log
      Shows the last five commits in your repository.

   pri | p NUMBER [A-X]
      Add priority specified (A, B, C, etc.) to item NUMBER.

   prepend | pre NUMBER "text to prepend"
      Add "text to prepend" to the beginning of the item.

   pull
      Pulls from your remote git repository.

   push
      Pushes to your remote git repository.

   status
      "git status" of the repository containing your todo files.
      Requires git version 1.7.4 or newer.
```

CHAPTER 2

Add-ons

There are two ways to write add-ons for `todo.py`.

1. Write an executable that works similar to the specifications for `todo.sh`.
2. Write a python module.

This documentation will cover the latter.

2.1 Writing a python module to extend `todo.py`.

Your module should start with a doc-string, e.g.,

```
"""
example_module
~~~~~~~~~~~~~~

My new awesome module to extend todo.py.

------

Author: Ian Cordasco

Commands:
    - foo | foobar
    - monty | montypython
    - spam
"""
```

You can put as much information in there as you like. It mainly helps other people and isn't used by `todo.py`.

After that, you should import at least one thing from `todo.py`:

- `usage` (introduced in v0.3.0)

This is a decorator which supplies some crucial information to `todo.py`. `usage` will tell `todo.py` what to print when a user runs `todo.py help`. An example implementation would be:

```
from todo import usage


@usage('\tfoo | foobar "Args if you want them"',
    '\t\tDescription of what `foo` does.')
def foo(*args):
```

```
    """Doc-string for foo()"""
    pass
```

2.2 Registering Your Commands

Unfortunately, the `command` decorator used internally in `todo.py` will not work properly for add-ons. Until I can design a working solution, you can either use the `command` decorator to construct your own dictionary of commands which `todo.py` will then use to recognize your script's commands.

Send any suggestions to graffatcolmingov [at] gmail or just send a pull request. (I'd rather the latter since it will more clearly attribute the idea to you.)