Contribute to Taskwarrior

Dirk Deimeke

Taskwarrior academy

FrOSCon 2016

Content

Dirk Deimeke (that's me)

- Born 1968 in Wanne-Eickel
- Linux since 1996
- Emigrated 2008 to Switzerland
- Taskwarrior Team since 2010

Entry point for more `https://d5e.org/`

Help is neeeded

Help is needed in all areas of Taskwarrior development – design, coding, translating, testing, support and marketing. Applicants must be friendly.

Perhaps you are looking to help, but don’t know where to start. You can of course email us but take a look at this list. Perhaps you have skills we are looking for, this slides show ways you may be able to help.

Topics (1)

- Use Taskwarrior, become familiar with it, and make suggestions. We get great feedback from both new users and veteran users. New users have a fresh approach that we can no longer achieve, while veteran users develop clever and crafty ways to use the product.
- Report bugs and odd behavior when you see it. We don’t necessarily know it’s broken, unless you tell us.
- Suggest enhancements. We get lots of these, and it’s great. Some really good ideas have been suggested and implemented. Sure, some are out of scope, or plain crazy, but the stream of suggestions is fascinating to think about.
- Participate in the bug tracking and the Q & A site, to help others and maybe learn something yourself.
- Help triage the issues list.
- Join the IRC channel #taskwarrior on freenode.net and help answer some questions.

Topics (2)

- Join either the developer-mailinglist or user-mailinglist (or both) and participate.
- Proofread the documentation and man pages.
- Improve the documentation.
- Improve the man pages.
- Help improve the tutorials. Make your own tutorial.
- Confirm a bug. Nothing gets fixed without confirmation.
- Refine a bug. Provide relevant details, elaborate on the behavior.
- Fix a bug. Send a patch. You'll need C++ skills for this.
- Write a unit test. Improve an existing unit test.
- Spread the word. Help others become more effective at managing tasks. Share your methodology, to inspire others.
- Encouragement. Tell us what works for you, and what doesn't. It's all good.
- Donate! Help offset costs.

Your skill is needed!

We need yuo!

Please remember that we need contributions from all skillsets, however small. **Every contribution helps.**

How to become an Open Source Contributor

There are many people who wish to start contributing, but don’t know how or where to start.

Because we want you to join in the fun with Open Source – it can be fun and rewarding, improve your skills, or just give you a way to contribute back to a project.

We’re going to pick the smallest contribution of all – a typo fix. While this may be a very small improvement, it is nevertheless a wanted improvement, and will be welcomed.

Fixes such as this happen many times a day. Similar work on new features, new documents, rewriting help, refactoring code, fixing bugs and improving performance all combine to make a project grow and improve.

Making a bigger change also is certainly an option, but the focus here is on going through the procedure, which is somewhat independent from the nature of the change.

Development Environment Setup

In order to build and test software, you need a development environment. That's just a term that means you need certain tools installed before proceeding. Here are the tools that Taskwarrior needs:

- Compiler: GCC 4.7 or newer, or Clang 3.4 or newer.
- Libraries: GnuTLS, and libuuid
- Tools: **Git**, CMake, make, Python

The procedure for installing this software is OS-dependent, but here are the commands you would use on Debian:

```
$ sudo apt-get install gcc
$ sudo apt-get install libgnutls28-dev
$ sudo apt-get install uuid-dev
$ sudo apt-get install git
$ sudo apt-get install cmake
$ sudo apt-get install make
```

Get the Code

Now you have the tools, next you need the code. This involves cloning the repository using git and looking at the development branch:

```
$ git clone https://git.tasktools.org/scm/tm/task.git task.git
Cloning into 'task.git'...
remote: Counting objects: 55345, done.
remote: Compressing objects: 100% (12655/12655), done.
remote: Total 55345 (delta 44868), reused 52340 (delta 42437)
Receiving objects: 100% (55345/55345), 25.04 MiB | 7.80 MiB/s, done.
Resolving deltas: 100% (44868/44868), done.
Checking connectivity... done.
```

The URL for the repository was obtained from looking around on https://git.tasktools.org where several repositories are public, including the one for the web site https://taskwarrior.org/.

Selecting the branch

Next you just need to get onto the right branch, where the development is happening. There are several branches, but fortunately it is easy to find the right one - it is the one with the highest number. Do this:

```
$ cd task.git
$ git branch -a
* master
  remotes/origin/2.5.1
  remotes/origin/2.5.2
  remotes/origin/2.6.0
  remotes/origin/HEAD -&gt; origin/master
  remotes/origin/master
$ git checkout 2.6.0
Branch 2.6.0 set up to track remote branch 2.6.0 from origin.
Switched to a new branch '2.6.0'
```

By the time you run this, things may have changed, and there could be a higher number than `2.6.0`. If that is the case, then use the higher version number instead.

Here's a thought – if the webpage does not show the latest branch names, then, you know, you could fix that.

Fix Something

Now that you have the code, find something to fix. This may be the hardest step, but knowing how many typos there are in the source code and docs, it shouldn't take long to find one. Try looking in the files in these directories:

- `task.git/doc/man`
- `task.git/scripts`
- `task.git/src`
- `task.git/test`

It also doesn't need to be a typo, it can instead be a poorly-worded sentence, or one that could be more clear. You'll find something, whether it is jargon, mixed tenses, mistakes, or just plain wrong.

Then fix it, using a text editor. Try to make the smallest possible change to achieve what you want, because smaller changes are easier to verify and approve, and no reviewer wants to receive a large change to approve.

Build Taskwarrior

Taskwarrior has an extensive test suite to prove that things are still working as expected. You'll need to build the program and run this test suite in order to prove to yourself that your fix is good. It may seem like building the program is overkill, if you only make a small change, but no, it is not. The test suite is there to save you from submitting a bad change, and to save Taskwarrior from any mistakes you make.

First you have to build the program. Do this:

```
$ cd task.git
$ cmake .
-- The C compiler identification is ...
...
-- Build files have been written to: /home/user/task.git
$ make
Scanning dependencies of target columns
...
[100%] Built target calc_executable
```

If the above commands worked, there will be a binary, which you can find with `ls -l src/task`.

Build and run the Test Suite

```
$ cd test
$ make
[ 14%] Built target task
...
[100%] Built target view.t
```

Now run the test suite, which can take anywhere from 10 - 500 seconds, depending on your hardware and OS:

```
$ ./run_all
Passed:                         8300
Failed:                            0
Unexpected successes:              0
Skipped:                           3
Expected failures:                 5
Runtime:                       32.50 seconds
```

We are looking for zero failed tests, as shown. This means all is well.

Commit the Change

Now you've made a change, built and tested the code. The next step is to commit the change locally. This example assumes you fixed a typo in the man page. Check to see which file you changed, stage that file, then commit it:

```
$ cd task.git
$ git status
On branch 2.6.0
Your branch is up-to-date with 'origin/2.6.0'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   doc/man/task.1.in

no changes added to commit (use "git add" and/or "git commit -a")
$ git add doc/man/task.1.in
$ git commit -m 'Docs: corrected typo in the main man page'
[2.6.0 ddbb07e] Docs: corrected typo in the main man page
 1 file changed, 1 insertion(+)
```

Notice how the commit message looks like this:
`Category: Brief description`, which is how the commit messages should look.

Make a Patch

Once the commit is made, making a patch is simple:

```
$ git format-patch HEAD^
0001-Docs-corrected-typo-in-the-main-man-page.patch
```

Submit the Patch

Finally you just need to email that patch file to our developer mailinglist taskwarrior-dev@googlegroups.com. You will need to attach it to an email, and not just paste it in, because the mail client will probably mess with the contents, wrapping lines etc, which can make it unusable.

What happens next is that a developer will take your patch and study it, to ascertain whether it really does fix something that is broken. If there is a problem, you'll hear back with some **gentle, constructive** criticism. If the problem is small, it might just get fixed. Then your patch is applied, tested, and if all looks well, pushed to the public repository, and included in the the next release. Your name will go into the `AUTHORS` file, and you will be thanked.

Congratulations!

Welcome to the wonderful world of open source involvement. Now do it again…

Coding Style – Golden Rule

The coding style used for the Taskwarrior, Taskserver, and other codebases is deliberately kept simple and a little vague. This is because there are many languages involved (C++, C, Python, Perl, sh, bash, HTML, troff and more), and specifying those would be a major effort that detracts from the main focus which is improving the software.

Instead, the general guideline is simply this:

Golden Rule

Make all changes and additions such that they blend in perfectly with the surrounding code, so it looks like only one person worked on the source, and that person is rigidly consistent.

Coding Style – Detailed

To be a little more explicit, the common elements across the languages are:

- Indent code using two spaces, no tabs
- With Python, follow PEP8 as much as possible
- Surround operators and expression terms with a space
- No cuddled braces
- Stick to 80 columns where possible, although exceptions are fine
- Class names are capitalized, variable names are not

How to report a bug

Please report bugs and odd behavior when you see it. We don't necessarily know it's broken, unless you tell us. A good bug description improves response time and reduces the burden on the developers.

It helps if you first determine whether this is a known bug. To do this, you will need to look at the current list of bugs here:

https://bug.tasktools.org

If you don't see the bug listed, sign up for an account at the link above, and create a new issue. Please include the following information, as it helps us categorize, prioritize and replicate the problem:

- Tell us *exactly* what command you ran
- Tell us what happened
- Tell us what you expected to happen

How to report a bug (2)

- Tell us about your installation, by running the command `task diagnostics` and including the output in the issue description. Notice that the `diagnostics` command was added specifically for reporting issues, and it gives us information about your operating system, libraries, compiler and so on. If you feel some of the information is personal, then remove that line.
- If you need to demonstrate a problem with a screenshot, you can hide your private data with the `rc.obfuscate1=` configuration override:

```
$ task 160-162,165 minimal

ID  Project Tags Description
160 tw.244       Activity report April 2015 [1]
162 tw.244   [1] Document new commit message standard
165 tw.244       Fix color.precedence regarding new priority UDA
161 tw.244       Insert step in CLI::analyze which restores quoted args based on heuristics

4 tasks
$ task 160-162,165 minimal rc.obfuscate=1

ID  Project Tags Description
160 xxxxxx       xxxxxxxx xxxxxx xxxxx xxxx xxx
162 xxxxxx   xxx xxxxxxxx xxx xxxxxx xxxxxxx xxxxxxxx
165 xxxxxx       xxx xxxxxxxxxxxxxxxx xxxxxxxxx xxx xxxxxxxx xxx
161 xxxxxx       xxxxxx xxxx xx xxxxxxxxxxxx xxxxx xxxxxxxx xxxxxx xxxx xxxxx xx xxxxxxxxxx

4 tasks
Configuration override rc.obfuscate:1
$
```

How to request a feature

We encourage you to request features.

You can submit a feature request using our issue tracking system. Simply create an account, create an issue, and set the type to either ’improvement’ or ’feature’, at your discretion. But there is a little more to it than that. After all, wouldn’t you like to see your request implemented?

What Makes a Good Feature Request?

Requesting a feature is easy, but you must remember that it doesn't mean it will get done. It may be a simple or vague request. It may be a complex request that needs some analysis, and we will want to discuss it first. Sometimes we even go so far as to write up an RFC before doing any work. Here are some tips for making a good feature request:

- Describe a compelling feature, and make us really *want* it. What we find most convincing is a statement that illustrates what having the new feature would mean, or would enable. What new things could you do if you had this feature? How would you be more effective at managing tasks with this feature?
- Describe how the feature might interact with existing features. We prefer it when a new feature is more than simply storing a new piece of data, for later retrieval – we have UDAs ("User defined attributes") for that.
- Describe any new reports that may be needed.

What Makes a Good Feature Request? (2)

- Describe who might benefit from this feature.
- Describe a feature that doesn't impose a cost on users that choose not to use it. We want the basic usage (add, list, done) to remain simple and effective.

One thing to bear in mind is that we have a long term goal of simplifying the way we manage tasks, simplifying Taskwarrior, and improving consistency.

Things to Avoid

- If your suggestion affects basic usage (add, list, done) then it will not be implemented.
- Don’t bother inventing syntax, or coming up with an unused character on your keyboard. We would much rather you put your effort into thinking about the feature and the benefits. Syntax is our problem. Focus on making others wonder how they could possibly live without the feature.
- Don’t bother suggesting features that lead Taskwarrior away from its intended purpose – managing tasks. We have no intention of adding time-tracking to the Taskwarrior core.
- If you have a suggestion that can be implemented as a hook script, or add-on, please do that instead. If the result is compelling enough, we may want to migrate the functionality in-core, but it’s not likely.
- Don’t tell us how easy something would be, or how quickly you could add it. Ease of implementation has nothing to do with whether an idea is good. Instead, convince everyone that the idea is great.

A Good Example

I think dependencies would be a great feature, because it would help me organize tasks and know which ones I need to work on first. It would also tell me which ones I cannot work on now. Tasks that are depended upon could affect the ’next’ report, because those tasks need to be done first. Ideally a task could depend on multiple other tasks. I would need to both add and remove a dependency. Dependencies are a commonly understood concept, and so would be useful to many users.

Another Good Example

I should be allowed to use ’status’ field in a report. It doesn’t make sense to withhold just this one field. For the sake of consistency, please add this as a reportable, sortable field.

A Bad Example

Why can’t we just make the @ character mean ……? How hard can that be?

Yes, we are seeking

Just for the glory, we are searching people taking over responsibilities and support us for a longer period of time.

The only thing reserved is overall feature and architecture limitations.

If you think you fit in, contact us, please.

Chief Bug Hunter

You *own* the bug databases for all projects. Manage the whole thing, reject the crazy, ask for more details, reproduce the problems, assign priorities and fix versions, tag the issues and so on.

Bugs need more love.

This is a major job opening we have.

Master of Testing

Software Quality Engineer – automated testing. Create new Python tests to increase test coverage (currently around 88%, although who knows), and improve release quality. Refactor old tests, add new ones, eliminate useless or redundant tests, speed up the tests. Create tests for newly-reported issues. Alert development to areas of weakness in the product, asymmetrical feature support, or newly occurring use cases.

Master of Picture and Sound

Demo Manager. We have used YouTube and Asciinema to great effect, because the products are somewhat visual, and everyone loves a live demo. But the demo scripts quickly get out of date, and new features are not shown. Own the tutorials, organize them, publish them, show off the features.

Maybe, you establish a visual changelog.

Chief Nudger

Technical community outreach. Own the relationships with package maintainers. Nudge them to update. Advise them not to package alpha/beta versions. Correct their docs. Identify distro gaps we need filled. Identify distros that are problematic for us: Debian and old GnuTLS libs, Debian refusing our PDF, Cygwin packaging issues, toolchain availability. Advise development. Pull back distro patches where appropriate.

Documentation Hero

Technical writer. Our online docs and man pages need organization, more examples, fewer examples, clearer descriptions, re-ordering, more screenshots, fewer screenshots, spell-checking, grammar-checking. We need a clear communicator with organization skills.

Head of Looking Nice Department

Website migration to static content generators. We need to evaluate, select and migrate to content generators using markdown text injected into our own templates. Need new templates.

What Have We Learned . . .

What Have We Learned From This Open Source Project?

Here is the collected wisdom that we have gained from running the Taskwarrior project for ten years. It has been rewarding, enjoyable, and sometimes frustrating. We learned a lot about users and Open Source expectations.

Advice To Open Source Project Contributors

Start an open source project if you want to learn all you can about software design, development, planning, testing, documenting, and delivery; enjoy technical challenges, administrative challenges, compromise, and will be satisfied hoping that someone out there is benefitting from your work.

Do not start an open source project **if you need praise, warmth and love** from your fellow human beings.

Features

If you could draw a boundary between that which is already supported, and that which is not, you would find that all the activity, discussion and drama occurs at that boundary.

Feature requests only nibble at the periphery. Bold changes originate elsewhere.

People will get excited about something a project doesn't yet support. Deliver it, and they will get excited about the next thing.

If a feature works well, you'll never hear about it again.

There is a fine line between *richly-featured* and *bloated*. There may not be a line at all.

If you demo two features, and talk about twenty more, users still only know about the two. Visual demonstrations have far greater impact.

Changes

Every change will ruin someone's day. They will be sure to tell you about it.

The same change will improve someone's day. You will not hear of this.

Many believe that if a change is small, it deserves to be in the project, regardless of whether it makes sense for it to be there.

Feature Requests

People will disguise feature requests as bugs, which means either they consider difference of opinion a defect, or believe that calling it a flaw will force implementation, but hopefully they just forgot to set the issue type to *enhancement*.

Attention!

Some people find it very difficult to articulate what they want. It's worth being patient and finding out what they need.

What you keep out of a project is just as important as what you allow in to a project.

Many new users will submit feature requests, just to show that they are knowledgeable and clever. They don't really want that feature, it's a form of positive feedback.

People

Beware of suggestions from users who have used your software for only a day or so. Be equally aware of suggestions from users who have used your software for a long, long time.

People will threaten to not use open source software because it lacks a feature, thereby mistaking themselves for paying customers.

Users will go to the effort of seeking you out online, to directly ask you a question that is answered two clicks from the front page of a website.

What have you tried so far? is the best question to identify time wasters.

People will pick a fight with you about all your incidental choices. Your issue tracker, your branching strategy, your version numbers, the text editor you use, and so on.

Documentation

A looping, animated GIF will be watched over and over, scrutinized and understood. A paragraph of text will likely be ignored.

Man pages are too densely crammed with information, and too lengthy, for most modern humans to ingest.

You can choose the most permissive software license, and people will still argue with you about your choice.

SEO consultants are not very good at searching the web, and learning that you operate an open source, non-profit project. It says a lot.

FOQ – Frequently Obnoxious Questions

We get many loaded, heavily biased questions, so here are some answers for our most Frequently Obnoxious Questions, grouped by theme.

WARNING!

Some of this in tongue-in-cheek. Most is not.

Misplaced Concerns About Our Infrastructure

Why don’t you just use Github?

- We have many repositories, many of them private, and all controlled by user keys.
- We do things with git hooks that Github does not permit.
- The Github issue tracker is . . . weak.
- We do use Github to mirror the Taskwarrior repository at https://github.com/taskwarrior/task.

Why is the website in HTML and not something like Markdown? Markdown would encourage more contributions.

- There have been sixteen website patches from the community in the last three years. None of those were new pages.

Misplaced Concerns About Our Infrastructure

Why all the Atlassian stuff like Jira, Stash, Confluence, Bamboo?

- Jira is great. Atlassian is very generous with open source licences.
- You think the GitHub issue tracker is the way to go? – We don't.
- Can you get the Source code of GitHub?

I don't like that you use a custom CI solution, instead of Travis. Why?

- We simultaneously test every commit, of every project, on seven platforms.
- Show us how Travis is better.

Would you guys consider moving off Google groups to a real mailing list?

- Sure if there was a good reason.

Why is our infrastructure important to you?

My Idea Is Wonderful

Why won't you support my syntactic sugar idea?

- Many suggestions for enhancements are just keystroke aliases for existing functionality, that can be achieved using shell functions and aliases.
- Show us your proof of concept using the above techniques. We'll show you that you have already solved your problem.

Why don't we make the % key mean *this thing*?

- We're not trying to make Taskwarrior more cryptic. We're trying to provide a simpler and more consistent interface.

Design Choices

Why doesn’t Taskwarrior just use DropBox or Git for syncing?

- Neither DropBox not Git merges tasks, you just think they do. Taskserver syncs and merges tasks.

Links

Most of the topics mentioned got shortened and are stolen from the website. They are worth a visit.

- Want to contribute?
- How to become an Open Source Contributor
- What Have We Learned From This Open Source Project?
- Coding Style
- How to Report a Bug
- How to Request a Feature

Thanks for your patience!

dirk@deimeke.net

d5e.org