Bugwarrior Documentation

Release 0.8.0

Ralph Bean

Sep 17, 2017

Contents

`bugwarrior` is a command line utility for updating your local taskwarrior database from your forge issue trackers.

CHAPTER 1

Build Status

Branch	Status
master	
develop	

CHAPTER 2

Contents

Getting bugwarrior

Requirements

To use bugwarrior, you need python 2.7 and taskwarrior. Upon installation, the setup script will automatically download and install missing python dependencies.

Note that some of those dependencies have a C extension module (e.g. the `cryptography` package). If those packages are not yet present on your system, the setup script will try to build them locally, for which you will need a C compiler (e.g. `gcc`) and the necessary header files (python and, for the cryptography package, openssl). A convenient way to install those is to use your usual package manager (`dnf`, `yum`, `apt`, etc). Header files are installed from development packages (e.g. `python-devel` and `openssl-devel` on Fedora or `python-dev` `libssl-dev` on Debian).

Installing from the Python Package Index

Installing it from http://pypi.python.org/pypi/bugwarrior is easy with **`pip`**:

```
$ pip install bugwarrior
```

Alternatively, you can use **`easy_install`** if you prefer:

```
$ easy_install bugwarrior
```

By default, `bugwarrior` will be installed with support for the following services: Bitbucket, Bugzilla, Github, Gitlab, Pagure, Phabricator, Redmine, Teamlab, Track and Versionone. There is optional support for Jira, Megaplan.ru and Active Collab but those require extra dependencies that are installed by specifying `bugwarrior[service]` in the commands above. For example, if you want to use bugwarrior with Jira:

```
$ pip install "bugwarrior[jira]"
```

Installing from Source

You can find the source on github at http://github.com/ralphbean/bugwarrior. Either fork/clone if you plan to do development on bugwarrior, or you can simply download the latest tarball:

```
$ wget https://github.com/ralphbean/bugwarrior/tarball/master -O bugwarrior-latest.
↪tar.gz
$ tar -xzvf bugwarrior-latest.tar.gz
$ cd ralphbean-bugwarrior-*
$ python setup.py install
```

Installing from Distribution Packages

bugwarrior has been packaged for Fedora. You can install it with the standard **dnf** (**yum**) package management tools as follows:

```
$ sudo dnf install bugwarrior
```

How to use

Just run `bugwarrior-pull`.

Cron

It's ideal to create a cron task like:

```
*/15 * * * *  /usr/bin/bugwarrior-pull
```

Bugwarrior can emit desktop notifications when it adds or completes issues to and from your local `~/.task/` db. If your `bugwarriorrc` file has notifications turned on, you'll also need to tell cron which display to use by adding the following to your crontab:

```
DISPLAY=:0
*/15 * * * *  /usr/bin/bugwarrior-pull
```

systemd timer

If you would prefer to use a systemd timer to run `bugwarrior-pull` on a schedule, you can create the following two files:

```
$ cat ~/.config/systemd/user/bugwarrior-pull.service
[Unit]
Description=bugwarrior-pull

[Service]
Environment="DISPLAY=:0"
ExecStart=/usr/bin/bugwarrior-pull
Type=oneshot

[Install]
WantedBy=default.target
```

```
$ cat ~/.config/systemd/user/bugwarrior-pull.timer
[Unit]
Description=Run bugwarrior-pull hourly and on boot

[Timer]
OnBootSec=15min
OnUnitActiveSec=1h

[Install]
WantedBy=timers.target
```

Once those files are in place, you can start and enable the timer:

```
$ systemctl --user enable bugwarrior-pull.timer
$ systemctl --user start bugwarrior-pull.timer
```

Exporting a list of UDAs

Most services define a set of UDAs in which bugwarrior store extra information about the incoming ticket. Usually, this includes things like the title of the ticket and its URL, but some services provide an extensive amount of metadata. See each service's documentation for more information.

For using this data in reports, it is recommended that you add these UDA definitions to your `taskrc` file. You can generate your list of UDA definitions by running the following command:

```
bugwarrior-uda
```

You can add those lines verbatim to your `taskrc` file if you would like Taskwarrior to know the human-readable name and data type for the defined UDAs.

Note: Not adding those lines to your `taskrc` file will have no negative effects aside from Taskwarrior not knowing the human-readable name for the field, but depending on what version of Taskwarrior you are using, it may prevent you from changing the values of those fields or using them in filter expressions.

How to Configure

First, add a file named `.config/bugwarrior/bugwarriorrc` to your home folder. This file must include at least a `[general]` section including the following option:

- `targets`: A comma-separated list of *other* section names to use as task sources.

Optional options include:

- `taskrc`: Specify which TaskRC configuration file to use. By default, will use the system default (usually `~/.taskrc`).
- `shorten`: Set to `True` to shorten links.
- `inline_links`: When `False`, links are appended as an annotation. Defaults to `True`.
- `annotation_links`: When `True` will include a link to the ticket as an annotation. Defaults to `False`.
- `annotation_comments`: When `False` skips putting issue comments into annotations. Defaults to `True`.

- `legacy_matching`: Set to `False` to instruct Bugwarrior to match issues using only the issue's unique identifiers (rather than matching on description).
- `log.level`: Set to one of `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`, or `DISABLED` to control the logging verbosity. By default, this is set to `DEBUG`.
- `log.file`: Set to the path at which you would like logging messages written. By default, logging messages will be written to stderr.
- `annotation_length`: Import maximally this number of characters of incoming annotations. Default: 45.
- `description_length`: Use maximally this number of characters in the description. Default: 35.
- `merge_annotations`: If `False`, bugwarrior won't bother with adding annotations to your tasks at all. Default: `True`.
- `merge_tags`: If `False`, bugwarrior won't bother with adding tags to your tasks at all. Default: `True`.
- `static_fields`: A comma separated list of attributes that shouldn't be *updated* by bugwarrior. Use for values that you want to tune manually. Default: `priority`.

In addition to the `[general]` section, sections may be named `[flavor.myflavor]` and may be selected using the `--flavor` option to `bugwarrior-pull`. This section will then be used rather than the `[general]` section.

A more-detailed example configuration can be found at *Example Configuration*.

Common Service Configuration Options

All services support common configuration options in addition to their service-specific features. These configuration options are meant to be prefixed with the service name, e.g. `github.add_tags`, or `gitlab.default_priority`.

The following options are supported:

- `SERVICE.only_if_assigned`: If set to a username, only import issues assigned to the specified user.
- `SERVICE.also_unassigned`: If set to `True` and `only_if_assigned` is set, then also create tasks for issues that are not assigned to anybody. Defaults to `False`.
- `SERVICE.default_priority`: Assign this priority ('L', 'M', or 'H') to newly-imported issues. Defaults to `M`.
- `SERVICE.add_tags`: A comma-separated list of tags to add to an issue. In most cases, plain strings will suffice, but you can also specify templates. See the section *Field Templates* for more information.

Field Templates

By default, Bugwarrior will import issues with a fairly verbose description template looking something like this:

```
(BW)Issue#10 - Fix perpetual motion machine .. http://media.giphy.com/media/
↪LldEzRPqyo2Yg/giphy.gif
```

but depending upon your workflow, the information presented may not be useful to you.

To help users build descriptions that suit their needs, all services allow one to specify a `SERVICE.description_template` configuration option, in which one can enter a one-line Jinja template. The context available includes all Taskwarrior fields and all UDAs (see section named 'Provided UDA Fields' for each service) defined for the relevant service.

Note: Jinja templates can be very complex. For more details about Jinja templates, please consult Jinja's Template Documentation.

For example, to pull-in Github issues assigned to @ralphbean, setting the issue description such that it is composed of only the Github issue number and title, you could create a service entry like this:

```
[ralphs_github_account]
service = github
github.username = ralphbean
github.description_template = {{githubnumber}}: {{githubtitle}}
```

You can also use this tool for altering the generated value of any other Taskwarrior record field by using the same kind of template.

Uppercasing the project name for imported issues:

```
SERVICE.project_template = {{project|upper}}
```

You can also use this feature to override the generated value of any field. This example causes imported issues to be assigned to the 'Office' project regardless of what project was assigned by the service itself:

```
SERVICE.project_template = Office
```

Password Management

You need not store your password in plain text in your *bugwarriorrc* file; you can enter the following values to control where to gather your password from:

password = @oracle:use_keyring Retrieve a password from the system keyring. The `bugwarrior-vault` command line tool can be used to manage your passwords as stored in your keyring (say to reset them or clear them). Extra dependencies must be installed with *pip install bugwarrior[keyring]* to enable this feature.

password = @oracle:ask_password Ask for a password at runtime.

password = @oracle:eval:<command> Use the output of <command> as the password. For instance, to integrate bugwarrior with the password manager pass you can use `@oracle:eval:pass my/password`.

Hooks

Use hooks to run commands prior to importing from bugwarrior-pull. bugwarrior-pull will run the commands in the order that they are specified below.

To use hooks, add a `[hooks]` section to your configuration, mapping the hook you'd like to use with a comma-separated list of scripts to execute.

```
[hooks]
pre_import = /home/someuser/backup.sh, /home/someuser/sometask.sh
```

Hook options:

- `pre_import`: The pre_import hook is invoked after all issues have been pulled from remote sources, but before they are synced to the TW db. If your pre_import script has a non-zero exit code, the `bugwarrior-pull` command will exit early.

Notifications

Add a `[notifications]` section to your configuration to receive notifications when a bugwarrior pull runs, and when issues are created, updated, or deleted by `bugwarrior-pull`:

```
[notifications]
notifications = True
backend = growlnotify
finished_querying_sticky = False
task_crud_sticky = True
only_on_new_tasks = True
```

Backend options:

Backend Name	Operating System	Required Python Modules
`growlnotify`	MacOS X	`gntp`
`gobject`	Linux	`gobject`

Note: The `finished_querying_sticky` and `task_crud_sticky` options have no effect if you are using a notification backend other than `growlnotify`.

Configuration files

bugwarrior will look at the following paths and read its configuration from the first existing file in this order:

- `~/.config/bugwarrior/bugwarriorrc`
- `~/.bugwarriorrc`
- `/etc/xdg/bugwarrior/bugwarriorrc`

The default paths can be altered using the environment variables *BUGWARRIORRC*, *XDG_CONFIG_HOME* and *XDG_CONFIG_DIRS*.

Environment Variables

BUGWARRIORRC

This overrides the default RC file.

XDG_CONFIG_HOME

By default, **bugwarrior** looks for a configuration file named `$XDG_CONFIG_HOME/bugwarrior/bugwarriorrc`. If `$XDG_CONFIG_HOME` is either not set or empty, a default equal to `$HOME/.config` is used.

XDG_CONFIG_DIRS

If it can't find a user-specific configuration file (either `$XDG_CONFIG_HOME/bugwarrior/bugwarriorrc` or `$HOME/.bugwarriorrc`), **bugwarrior** looks through the directories in `$XDG_CONFIG_DIRS` for a configuration file named `bugwarrior/bugwarriorrc`. The directories in `$XDG_CONFIG_DIRS` should be separated with a colon ':'. If `$XDG_CONFIG_DIRS` is either not set or empty, a value equal to `/etc/xdg` is used.

Supported Services

Bugwarrior currently supports the following services:

ActiveCollab 4

You can import tasks from your activeCollab 4.x instance using the `activecollab` service name.

Additional Requirements

Install the following packages using `pip`:

- `pypandoc`
- `pyac`

Instructions

Obtain your user ID and API url by logging in, clicking on your avatar on the lower left-hand of the page. When on that page, look at the URL. The number that appears after "/user/" is your user ID.

On the same page, go to Options and API Subscriptions. Generate a read-only API key and add that to your bugwarriorrc file.

Bugwarrior will gather tasks and subtasks returned from the *my-tasks* API call. Additional API calls will be made to gather comments associated with each task.

Note: Use of the ActiveCollab service requires that the following additional python modules be installed.

- pypandoc
- pyac

Example Service

Here's an example of an activecollab target. This is only valid for activeCollab 4.x and greater, see *ActiveCollab 2* for activeCollab2.x.

```
[my_bug_tracker]
service = activecollab
activecollab.url = https://ac.example.org/api.php
activecollab.key = your-api-key
activecollab.user_id = 15
```

The above example is the minimum required to import issues from ActiveCollab 4. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Provided UDA Fields

Field Name	Description	Type
acbody	Body	Text (string)
accreatedbyname	Created By Name	Text (string)
accreatedon	Created On	Date & Time
acid	ID	Text (string)
acname	Name	Text (string)
acpermalink	Permalink	Text (string)
acprojectid	Project ID	Text (string)
actaskid	Task ID	Text (string)
actype	Task Type	Text (string)
acestimatedtime	Estimated Time	Text (numeric)
actrackedtime	Tracked Time	Text (numeric)
acmilestone	Milestone	Text (string)

ActiveCollab 2

You can import tasks from your ActiveCollab2 instance using the `activecollab2` service name.

Instructions

You can obtain your user ID and API url by logging into ActiveCollab and clicking on "Profile" and then "API Settings". When on that page, look at the URL. The integer that appears after "/user/" is your user ID.

Projects should be entered in a comma-separated list, with the project id as the key and the name you'd like to use for the project in Taskwarrior entered as the value. For example, if the project ID is 8 and the project's name in ActiveCollab is "Amazing Website" then you might enter 8:amazing_website

Note that due to limitations in the ActiveCollab API, there is no simple way to get a list of all tasks you are responsible for in AC. Instead you need to look at each ticket that you are subscribed to and check to see if your user ID is responsible for the ticket/task. What this means is that if you have 5 projects you want to query and each project has 20 tickets, you'll make 100 API requests each time you run `bugwarrior-pull`.

Example Service

Here's an example of an activecollab2 target. Note that this will only work with ActiveCollab 2.x - see above for 3.x and greater.

```
[my_bug_tracker]
services = activecollab2
activecollab2.url = http://ac.example.org/api.php
activecollab2.key = your-api-key
activecollab2.user_id = 15
activecollab2.projects = 1:first_project, 5:another_project
```

The above example is the minimum required to import issues from ActiveCollab 2. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Provided UDA Fields

Field Name	Description	Type
ac2body	Body	Text (string)
ac2createdbyid	Created By	Text (string)
ac2createdon	Created On	Date & Time
ac2name	Name	Text (string)
ac2permalink	Permalink	Text (string)
ac2projectid	Project ID	Text (string)
ac2ticketid	Ticket ID	Text (string)
ac2type	Task Type	Text (string)

Bitbucket

You can import tasks from your Bitbucket instance using the `bitbucket` service name.

Example Service

Here's an example of a Bitbucket target:

```
[my_issue_tracker]
service = bitbucket
bitbucket.username = ralphbean
bitbucket.login = ralphbean
bitbucket.password = mypassword
```

The above example is the minimum required to import issues from Bitbucket. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Note that both `bitbucket.username` and `bitbucket.login` are required and can be set to different values. `bitbucket.login` is used to specify what account bugwarrior should use to login to bitbucket. `bitbucket.username` indicates which repositories should be scraped. For instance, I always have `bitbucket.login` set to ralphbean (my account). But I have some targets with `bitbucket.username` pointed at organizations or other users to watch issues there.

As an alternative to password authentication, there is OAuth. To get a key and secret, go to the "OAuth" section of your profile settings and click "Add consumer". Set the "Callback URL" to `https://localhost/` and set the appropriate permissions. Then assign your consumer's credentials to `bitbucket.key` and `bitbucket.secret`. Note that you will have to provide a password (only) the first time you pull, so you may want to set `bitbucket.password = @oracle:ask_password` and run `bugwarrior-pull --interactive` on your next pull.

Service Features

Include and Exclude Certain Repositories

If you happen to be working with a large number of projects, you may want to pull issues from only a subset of your repositories. To do that, you can use the `bitbucket.include_repos` option.

For example, if you would like to only pull-in issues from your `project_foo` and `project_fox` repositories, you could add this line to your service configuration:

```
bitbucket.include_repos = project_foo,project_fox
```

Alternatively, if you have a particularly noisy repository, you can instead choose to import all issues excepting it using the `bitbucket.exclude_repos` configuration option.

In this example, `noisy_repository` is the repository you would *not* like issues created for:

```
bitbucket.exclude_repos = noisy_repository
```

Please note that the API returns all lowercase names regardless of the case of the repository in the web interface.

Filter Merge Requests

Although you can filter issues using *Common Service Configuration Options*, pull requests are not filtered by default. You can filter pull requests by adding the following configuration option:

```
bitbucket.filter_merge_requests = True
```

Provided UDA Fields

Field Name	Description	Type
`bitbucketid`	Issue ID	Text (string)
`bitbuckettitle`	Title	Text (string)
`bitbucketurl`	URL	Text (string)

Debian Bug Tracking System (BTS)

You can import tasks from the Debian Bug Tracking System (BTS) using the `bts` service name. Debian's bugs are public and no authentication information is required by bugwarrior for this service.

Additional Requirements

You will need to install the following additional packages via `pip`:

- `PySimpleSOAP`
- `python-debianbts`

Note: If you have installed the Debian package for bugwarrior, this dependency will already be satisfied.

Example Service

Here's an example of a Debian BTS target:

```
[debian_bts]
service = bts
bts.email = username@debian.org
```

The above example is the minimum required to import issues from the Debian BTS. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

Service Features

Include all bugs for packages

If you would like more bugs than just those you are the owner of, you can specify the `bts.packages` option.

For example if you wanted to include bugs on the `hello` package, you can add this line to your service configuration:

```
bts.packages = hello
```

More packages can be specified seperated by commas.

Ultimate Debian Database (UDD) Bugs Search

If you maintain a large number of packages and you wish to include bugs from all packages where you are listed as a Maintainer or an Uploader in the Debian archive, you can enable the use of the UDD Bugs Search.

This will peform a search and include the bugs from the result. To enable this feature, you can add this line to your service configuration:

```
bts.udd = True
```

Excluding bugs marked pending

Debian bugs are not considered closed until the fixed package is present in the Debian archive. Bugs do cease to be outstanding tasks however as soon as you have completed the work, and so it can be useful to exclude bugs that you have marked with the pending tag in the BTS.

This is the default behaviour, but if you feel you would like to include bugs that are marked as pending in the BTS, you can disable this by adding this line to your service configuration:

```
bts.ignore_pending = False
```

Excluding sponsored and NMU'd packages

If you maintain an even larger number of packages, you may wish to exclude some packages.

You can exclude packages that you have sponsored or have uploaded as a non-maintainer upload or team upload by adding the following line to your service configuration:

```
bts.udd_ignore_sponsor = True
```

Note: This will only affect the bugs returned by the UDD bugs search service and will not exclude bugs that are discovered due to ownership or due to packages explicitly specified in the service configuration.

Excluding packages explicitly

If you would like to exclude a particularly noisy package, that is perhaps team maintained, or a package that you have orphaned and no longer have interest in but are still listed as Maintainer or Uploader in stable suites, you can explicitly

ignore bugs based on their binary or source package names. To do this add one of the following lines to your service configuration:

```
bts.ignore_pkg = hello,anarchism
bts.ignore_src = linux
```

Note: The `src:` prefix that is commonly seen in the Debian BTS interface is not required when specifying source packages to exclude.

Provided UDA Fields

Field Name	Description	Type
`btsnumber`	Bug Number	Text (string)
`btsurl`	bugs.d.o URL	Text (string)
`btssubject`	Subject	Text (string)
`btssource`	Source Package	Text (string)
`btspackage`	Binary Package	Text (string)
`btsforwarded`	Forwarded URL	Text (string)
`btsstatus`	Status	Text (string)

Bugzilla

You can import tasks from your Bz instance using the `bugzilla` service name.

Additional Dependencies

Install packages needed for Bugzilla support with:

```
pip install bugwarrior[bugzilla]
```

Example Service

Here's an example of a bugzilla target.

This will scrape every ticket that:

1. Is not closed and
2. rbean@redhat.com is either the owner, reporter or is cc'd on the issue.

Bugzilla instances can be quite different from one another so use this with caution and please report bugs so we can make bugwarrior support more robust!

```
[my_issue_tracker]
service = bugzilla
bugzilla.base_uri = bugzilla.redhat.com
bugzilla.username = rbean@redhat.com
bugzilla.password = OMG_LULZ
```

Alternately, if you are using a version of python-bugzilla newer than 2.1.0, you can specify an API key instead of a password. Note that the username is still required in this case, in order to identify bugs belonging to you.

```
bugzilla.api_key = 4f4d475f4c554c5a4f4d475f4c554c5a
```

The above example is the minimum required to import issues from Bugzilla. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*. Note, however, that the filtering options, including `only_if_assigned` and `also_unassigned`, do not work

There is an option to ignore bugs that you are only cc'd on:

```
bugzilla.ignore_cc = True
```

But this will continue to include bugs that you reported, regardless of whether they are assigned to you.

If your bugzilla "actionable" bugs only include ON_QA, FAILS_QA, PASSES_QA, and POST:

```
bugzilla.open_statuses = ON_QA,FAILS_QA,PASSES_QA,POST
```

This won't create tasks for bugs in other states. The default open statuses: "NEW,ASSIGNED,NEEDINFO,ON_DEV,MODIFIED,POST,REOPENED,ON_QA,FAILS_QA,PASSES_QA"

If you're on a more recent Bugzilla install, the NEEDINFO status no longer exists, and has been replaced by the "needinfo?" flag. Set "bugzilla.include_needinfos" to "True" to have taskwarrior also add bugs where information is requested of you. The "bugzillaneedinfo" UDA will be filled in with the date the needinfo was set.

To see all your needinfo bugs, you can use "task bugzillaneedinfo.any: list".

If the filtering options are not sufficient to find the set of bugs you'd like, you can tell Bugwarrior exactly which bugs to sync by pasting a full query URL from your browser into the `bugzilla.query_url` option:

```
bugzilla.query_url = https://bugzilla.mozilla.org/query.cgi?bug_status=ASSIGNED&
↪email1=myname%40mozilla.com&emailassigned_to1=1&emailtype1=exact
```

Provided UDA Fields

Field Name	Description	Type
`bugzillasummary`	Summary	Text (string)
`bugzillaurl`	URL	Text (string)
`bugzillabugid`	Bug ID	Numeric (integer)
`bugzillastatus`	Bugzilla Status	Text (string)
`bugzillaneedinfo`	Needinfo	Date

Gerrit

You can import code reviews from a Gerrit instance using the `gerrit` service name.

Example Service

Here's an example of a gerrit project:

```
[my_issue_tracker]
service = gerrit
gerrit.base_uri = https://yourhomebase.xyz/gerrit/
gerrit.username = your_username
gerrit.password = your_http_digest_password
```

The above example is the minimum required to import issues from Gerrit.

Note that the password is typically not your normal login password. Go to the "HTTP Password" section in your account settings to generate/retrieve this password.

You can also pass an optional `gerrit.ssl_ca_path` option which will use an alternative certificate authority to verify the connection.

You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Provided UDA Fields

Field Name	Description	Type
`gerritid`	Issue ID	Text (string)
`gerritsummary`	Summary	Text (string)
`gerriturl`	URL	Text (string)

The Gerrit service provides a limited set of UDAs. If you have need for some other values not present here, please file a request (there's lots of metadata in there that we could expose).

Github

You can import tasks from your Github instance using the `github` service name.

Example Service

Here's an example of a Github target:

```
[my_issue_tracker]
service = github
github.login = ralphbean
github.password = OMG_LULZ
github.username = ralphbean
```

The above example is the minimum required to import issues from Github. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

`github.login` is used to specify what account bugwarrior should use to login to github, combined with `github.password`.

If two-factor authentication is used, `github.token` must be given rather than `github.password`. To get a token, go to the "Personal access tokens" section of your profile settings. Only the `public_repo` scope is required, but access to private repos can be gained with `repo` as well.

Service Features

Repo Owner

`github.username` indicates which repositories should be scraped. For instance, I always have `github.login` set to ralphbean (my account). But I have some targets with `github.username` pointed at organizations or other users to watch issues there. This parameter is required unless `github.query` is provided.

Include and Exclude Certain Repositories

By default, issues from all repos belonging to `github.username` are included. To turn this off, set:

```
github.include_user_repos = False
```

If you happen to be working with a large number of projects, you may want to pull issues from only a subset of your repositories. To do that, you can use the `github.include_repos` option.

For example, if you would like to only pull-in issues from your `project_foo` and `project_fox` repositories, you could add this line to your service configuration:

```
github.include_repos = project_foo,project_fox
```

Alternatively, if you have a particularly noisy repository, you can instead choose to import all issues excepting it using the `github.exclude_repos` configuration option.

In this example, `noisy_repository` is the repository you would *not* like issues created for:

```
github.exclude_repos = noisy_repository
```

Import Labels as Tags

The Github issue tracker allows you to attach labels to issues; to use those labels as tags, you can use the `github.import_labels_as_tags` option:

```
github.import_labels_as_tags = True
```

Also, if you would like to control how these labels are created, you can specify a template used for converting the Github label into a Taskwarrior tag.

For example, to prefix all incoming labels with the string '*github*' (perhaps to differentiate them from any existing tags you might have), you could add the following configuration option:

```
github.label_template = github_{{label}}
```

In addition to the context variable `{{label}}`, you also have access to all fields on the Taskwarrior task if needed.

Note: See *Field Templates* for more details regarding how templates are processed.

Filter Pull Requests

Although you can filter issues using *Common Service Configuration Options*, pull requests are not filtered by default. You can filter pull requests by adding the following configuration option:

```
github.filter_pull_requests = True
```

Get involved issues

By default, bugwarrior pulls all issues across owned and member repositories assigned to the authenticated user. To disable this behavior, use:

```
github.include_user_issues = False
```

Instead of fetching issues and pull requests based on `{{username}}`'s owned repositories, you may instead get those that `{{username}}` is involved in. This includes all issues and pull requests where the user is the author, the assignee, mentioned in, or has commented on. To do so, add the following configuration option:

```
github.involved_issues = True
```

Queries

If you want to write your own github query, as described at https://help.github.com/articles/searching-issues/:

```
github.query = assignee:octocat is:open
```

Note that this search covers both issues and pull requests, which github treats as a special kind of issue.

To disable the pre-defined queries described above and synchronize only the issues matched by the query, set:

```
github.include_user_issues = False
github.include_user_repos = False
```

GitHub Enterprise Instance

If you're using GitHub Enterprise, the on-premises version of GitHub, you can point bugwarrior to it with the `github.host` configuration option. E.g.:

```
github.host = github.acme.biz
```

Provided UDA Fields

Field Name	Description	Type
`githubbody`	Body	Text (string)
`githubcreatedon`	Created	Date & Time
`githubmilestone`	Milestone	Text (string)
`githubnumber`	Issue/PR #	Numeric
`githubtitle`	Title	Text (string)
`githubtype`	Type	Text (string)
`githubupdatedat`	Updated	Date & Time
`githuburl`	URL	Text (string)
`githubrepo`	username/reponame	Text (string)
`githubuser`	Author of issue/PR	Text (string)

Gitlab

You can import tasks from your Gitlab instance using the `gitlab` service name.

Example Service

Here's an example of a Gitlab target:

```
[my_issue_tracker]
service = gitlab
gitlab.login = ralphbean
gitlab.token = OMG_LULZ
gitlab.host = gitlab.com
```

The above example is the minimum required to import issues from Gitlab. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

The `gitlab.token` is your private API token.

Service Features

Include and Exclude Certain Repositories

If you happen to be working with a large number of projects, you may want to pull issues from only a subset of your repositories. To do that, you can use the `gitlab.include_repos` option.

For example, if you would like to only pull-in issues from your own `project_foo` and team `bar`'s `project_fox` repositories, you could add this line to your service configuration (replacing `me` by your own login):

```
gitlab.include_repos = me/project_foo, bar/project_fox
```

Alternatively, if you have a particularly noisy repository, you can instead choose to import all issues excepting it using the `gitlab.exclude_repos` configuration option.

In this example, `noisy/repository` is the repository you would *not* like issues created for:

```
gitlab.exclude_repos = noisy/repository
```

Hint: If you omit the repository's namespace, bugwarrior will automatically add your login as namespace. E.g. the following are equivalent:

```
gitlab.login = foo
gitlab.include_repos = bar
```

and:

```
gitlab.login = foo
gitlab.include_repos = foo/bar
```

Filtering Repositories with Regular Expressions

If you don't want to list every single repository you want to include or exclude, you can additionally use the options `gitlab.include_regex` and `gitlab.exclude_regex` and specify a regular expression (suitable for Python's `re` module). No default namespace is applied here, the regular expressions are matched to the full repository name with its namespace.

The regular expressions can be used in addition to the lists explained above. So if a repository is not included in `gitlab.include_repos`, it can still be included by `gitlab.include_regex`, and vice versa; and likewise for `gitlab.exclude_repos` and `gitlab.exclude_regex`.

Note: If a repository matches both the inclusion and the exclusion options, the exclusion takes precedence.

For example, you want to include only the repositories `foo/node` and `bar/node` as well as all repositories in the namespace `foo` starting with `ep_`, but not `foo/ep_example`:

```
gitlab.include_repos = foo/node, bar/node
gitlab.include_regex = foo/ep_.*
gitlab.exclude_repos = foo/ep_example
```

Import Labels as Tags

The gitlab issue tracker allows you to attach labels to issues; to use those labels as tags, you can use the `gitlab.import_labels_as_tags` option:

```
gitlab.import_labels_as_tags = True
```

Also, if you would like to control how these labels are created, you can specify a template used for converting the gitlab label into a Taskwarrior tag.

For example, to prefix all incoming labels with the string '*gitlab*' (perhaps to differentiate them from any existing tags you might have), you could add the following configuration option:

```
gitlab.label_template = gitlab_{{label}}
```

In addition to the context variable `{{label}}`, you also have access to all fields on the Taskwarrior task if needed.

Note: See *Field Templates* for more details regarding how templates are processed.

Include Merge Requests

Although you can filter issues using *Common Service Configuration Options*, merge requests are not filtered by default. You can filter merge requests by adding the following configuration option:

```
gitlab.filter_merge_requests = True
```

Include Todo Items

By default todo items are not included. You may include them by adding the following configuration option:

```
gitlab.include_todos = True
```

If todo items are included, by default, todo items for all projects are included. To only fetch todo items for projects which are being fetched, you may set:

```
gitlab.include_all_todos = False
```

Include Only One Author

If you would like to only pull issues and MRs that you've authored, you may set:

```
gitlab.only_if_author = myusername
```

Use HTTP

If your Gitlab instance is only available over HTTP, set:

```
gitlab.use_https = False
```

Do Not Verify SSL Certificate

If you want to ignore verifying the SSL certificate, set:

```
gitlab.verify_ssl = False
```

Provided UDA Fields

Field Name	Description	Type
`gitlabdescription`	Description	Text (string)
`gitlabcreatedon`	Created	Date & Time
`gitlabmilestone`	Milestone	Text (string)
`gitlabnumber`	Issue/MR #	Numeric
`gitlabtitle`	Title	Text (string)
`gitlabtype`	Type	Text (string)
`gitlabupdatedat`	Updated	Date & Time
`gitlabduedate`	Due Date	Date
`gitlaburl`	URL	Text (string)
`gitlabrepo`	username/reponame	Text (string)
`gitlabupvotes`	Number of upvotes	Numeric
`gitlabdownvotes`	Number of downvotes	Numeric
`gitlabwip`	Work-in-Progress flag	Numeric
`gitlabauthor`	Issue/MR author	Text (string)
`gitlabassignee`	Issue/MR assignee	Text (string)
`gitlabnamespace`	project namespace	Text (string)

Gmail

You can create tasks from e-mails in your Gmail account using the `gmail` service name.

Additional Dependencies

Install packages needed for Gmail support with:

:: pip install bugwarrior[gmail]

Client Secret

In order to use this service, you need to create a product and download a client secret file. Do this by following the instructions on: `https://developers.google.com/gmail/api/quickstart/python`. You should save the resulting secret in your home directory as `.gmail_client_secret.json`. You can override this location by setting the `client_secret_path` option.

Example Service

Here's an example of a gmail target:

:: [my_gmail] service = gmail gmail.query = label:action OR label:readme gmail.login_name = you@example.com

The specified query can be any gmail search term. By default it will select starred threads. One task is created per selected thread, not per e-mail.

You do not need to specify the `login_name`, but it can be useful to avoid accidentally fetching data from the wrong account. (This also allows multiple targets with the same login to share the same authentication token.)

Authentication

When you first run `bugwarrior-pull`, a browser will be opened and you'll be asked to authorise the application to access your e-mail. Once authorised a token will be stored in your bugwarrior data directory.

Provided UDA Fields

+———————+———————————+———————+| | `gmailthreadid` | Thread Id | Text (string) | +———————+———————————+———————+| | `gmailsubject` | Subject | Text (string) | +———————+———————————+———————+| | `gmailurl` | URL | Text (string) | +———————+———————————+———————+| | `gmaillastsender` | Last Sender's Name | Text (string) | +———————+———————————+———————+| | `gmaillastsender` | Last Sender's E-mail Address | Text (string) | +———————+———————————+———————+| | `gmailsnippet` | Snippet of text from conversation | Text (string) | +———————+———————————+———————+

Jira

You can import tasks from your Jira instance using the `jira` service name.

Additional Requirements

Install the following package using `pip`:

- `jira`

Example Service

Here's an example of a jira project:

```
[my_issue_tracker]
service = jira
jira.base_uri = https://bug.tasktools.org
jira.username = ralph
jira.password = OMG_LULZ
```

Note: The `base_uri` must not have a trailing slash.

The above example is the minimum required to import issues from Jira. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

Service Features

Specify the Query to Use for Gathering Issues

By default, the JIRA plugin will include any issues that are assigned to you but do not yet have a resolution set, but you can fine-tune the query used for gathering issues by setting the `jira.query` parameter.

For example, to select issues assigned to 'ralph' having a status that is not 'closed' and is not 'resolved', you could add the following configuration option:

```
jira.query = assignee = ralph and status != closed and status != resolved
```

This query needs to be modified accordingly to the literal values of your Jira instance; if the name contains any character, just put it in quotes, e.g.

> jira.query = assignee = 'firstname.lastname' and status != Closed and status != Resolved and status != Done

Jira v4 Support

If you happen to be using a very old version of Jira, add the following configuration option to your service configuration:

```
jira.version = 4
```

Do Not Verify SSL Certificate

If you want to ignore verifying the SSL certificate, set:

```
jira.verify_ssl = False
```

Import Labels and Sprints as Tags

The Jira issue tracker allows you to attach labels to issues; to use those labels as tags, you can use the `jira.import_labels_as_tags` option:

```
jira.import_labels_as_tags = True
```

You can also import the names of any sprints associated with an issue as tags, by setting the `jira.import_sprints_as_tags` option:

```
jira.import_sprints_as_tags = True
```

If you would like to control how these labels are created, you can specify a template used for converting the Jira label into a Taskwarrior tag.

For example, to prefix all incoming labels with the string '*jira*' (perhaps to differentiate them from any existing tags you might have), you could add the following configuration option:

```
jira.label_template = jira_{{label}}
```

In addition to the context variable `{{label}}`, you also have access to all fields on the Taskwarrior task if needed.

Note: See *Field Templates* for more details regarding how templates are processed.

Kerberos authentication

If the `password` is specified as `@kerberos`, the service plugin will try to authenticate against server with kerberos. A ticket must be already present on the client (created by running `kinit` or any other method).

Provided UDA Fields

Field Name	Description	Type
`jiradescription`	Description	Text (string)
`jiraid`	Issue ID	Text (string)
`jirasummary`	Summary	Text (string)
`jiraurl`	URL	Text (string)
`jiraestimate`	Estimate	Decimal (numeric)

Megaplan

You can import tasks from your Megaplan instance using the `megaplan` service name.

Additional Requirements

Install the following package using `pip`:

- `megaplan`

Example Service

Here's an example of a Megaplan target:

```
[my_issue_tracker]
service = megaplan
megaplan.hostname = example.megaplan.ru
megaplan.login = alice
```

```
megaplan.password = secret
megaplan.project_name = example
```

The above example is the minimum required to import issues from Megaplab. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Provided UDA Fields

Field Name	Description	Type
megaplanid	Issue ID	Text (string)
megaplantitle	Title	Text (string)
megaplanurl	URL	Text (string)

Pagure

You can import tasks from your private or public pagure instance using the `pagure` service name.

Example Service

Here's an example of a Pagure target:

```
[my_issue_tracker]
service = pagure
pagure.tag = releng
pagure.base_url = https://pagure.io
```

The above example is the minimum required to import issues from Pagure. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

Note that **either** `pagure.tag` or `pagure.repo` is required.

- `pagure.tag` offers a flexible way to import issues from many pagure repos. It will include issues from *every* repo on the pagure instance that is *tagged* with the specified tag. It is similar in usage to a github "organization". In the example above, the entry will pull issues from all "releng" pagure repos.
- `pagure.repo` offers a simple way to import issues from a single pagure repo.

Note – no authentication tokens are needed to pull issues from pagure.

Service Features

Include and Exclude Certain Repositories

If you happen to be working with a large number of projects, you may want to pull issues from only a subset of your repositories. To do that, you can use the `pagure.include_repos` option.

For example, if you would like to only pull-in issues from your `project_foo` and `project_fox` repositories, you could add this line to your service configuration:

```
pagure.tag = fedora-infra
pagure.include_repos = project_foo,project_fox
```

Alternatively, if you have a particularly noisy repository, you can instead choose to import all issues excepting it using the `pagure.exclude_repos` configuration option.

In this example, `noisy_repository` is the repository you would *not* like issues created for:

```
pagure.tag = fedora-infra
pagure.exclude_repos = noisy_repository
```

Import Labels as Tags

The Pagure issue tracker allows you to attach tags to issues; to use those pagure tags as taskwarrior tags, you can use the `pagure.import_tags` option:

```
pagure.import_tags = True
```

Also, if you would like to control how these taskwarrior tags are created, you can specify a template used for converting the Pagure tag into a Taskwarrior tag.

For example, to prefix all incoming labels with the string 'pagure' (perhaps to differentiate them from any existing tags you might have), you could add the following configuration option:

```
pagure.label_template = pagure_{{label}}
```

In addition to the context variable `{{label}}`, you also have access to all fields on the Taskwarrior task if needed.

Note: See *Field Templates* for more details regarding how templates are processed.

Provided UDA Fields

Field Name	Description	Type
`paguredatecreated`	Created	Date & Time
`pagurenumber`	Issue/PR #	Numeric
`paguretitle`	Title	Text (string)
`paguretype`	Type	Text (string)
`pagureurl`	URL	Text (string)
`pagurerepo`	username/reponame	Text (string)

Phabricator

You can import tasks from your Phabricator instance using the `phabricator` service name.

Additional Requirements

Install the following package using `pip`:

- `phabricator`

Example Service

Here's an example of a Phabricator target:

```
[my_issue_tracker]
service = phabricator
```

Note: Although this may not look like enough information for us to gather information from Phabricator, but credentials will be gathered from the user's `~/.arcrc`.

The above example is the minimum required to import issues from Phabricator. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Service Features

If you have dozens of users and projects, you might want to pull the tasks and code review requests only for the specific ones.

If you want to show only the tasks related to a specific user, you just need to add its PHID to the service configuration like this:

```
phabricator.user_phids = PHID-USER-ab12c3defghi45jkl678
```

If you want to show only the tasks and diffs related to a specific project or a repository, just add their PHIDs to the service configuration:

```
phabricator.project_phids = PHID-PROJ-ab12c3defghi45jkl678,PHID-REPO-
↪ab12c3defghi45jkl678
```

Both `phabricator.user_phids` and `phabricator.project_phids` accept a comma-separated (no spaces) list of PHIDs.

If you specify both, you will get tasks and diffs that match one **or** the other.

When working on a Phabricator installations with a huge number of users or projects, it is recommended that you specify `phabricator.user_phids` and/or `phabricator.project_phids`, as the Phabricator API may return a timeout for a query with too many results.

If you do not know PHID of a user, project or repository, you can find it out by querying Phabricator Conduit (`https://YOUR_PHABRICATOR_HOST/conduit/`) – the methods which return the needed info are `user.query`, `project.query` and `repository.query` respectively.

Provided UDA Fields

Field Name	Description	Type
`phabricatorid`	Object	Text (string)
`phabricatortitle`	Title	Text (string)
`phabricatortype`	Type	Text (string)
`phabricatorurl`	URL	Text (string)

Redmine

You can import tasks from your Redmine instance using the `redmine` service name.

Only first 100 issues are imported at the moment.

Example Service

Here's an example of a Redmine target:

```
[my_issue_tracker]
service = redmine
redmine.url = http://redmine.example.org/
redmine.key = c0c4c014cafebabe
redmine.user_id = 7
redmine.project_name = redmine
redmine.issue_limit = 100
```

You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

There are also *redmine.login*/*redmine.password* settings if your instance is behind basic auth.

Provided UDA Fields

Field Name	Description	Type
redmineid	ID	Text (string)
redminesubject	Subject	Text (string)
redmineurl	URL	Text (string)

Taiga

You can import tasks from a Taiga instance using the `taiga` service name.

Example Service

Here's an example of a taiga project:

```
[my_issue_tracker]
service = taiga
taiga.base_uri = http://taiga.fedorainfracloud.org
taiga.auth_token = ayJ1c4VyX2F1dGhlbnQpY2F0aW9uX2lmIjo1fQ:2a2LPT:qscLbfQC_
↪jyejQsICET5KgYNPLM
```

The above example is the minimum required to import issues from Taiga. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Service Features

By default, userstories from taiga are added in taskwarrior. If you like to include taiga tasks as well, set the config option:

```
taiga.include_tasks = True
```

Provided UDA Fields

Field Name	Description	Type
taigaid	Issue ID	Text (string)
taigasummary	Summary	Text (string)
taigaurl	URL	Text (string)

The Taiga service provides a limited set of UDAs. If you have need for some other values not present here, please file a request (there's lots of metadata in there that we could expose).

Teamlab

You can import tasks from your Teamlab instance using the `teamlab` service name.

Example Service

Here's an example of a Teamlab target:

```
[my_issue_tracker]
service = teamlab
teamlab.hostname = teamlab.example.com
teamlab.login = alice
teamlab.password = secret
teamlab.project_name = example_teamlab
```

The above example is the minimum required to import issues from Teamlab. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Provided UDA Fields

Field Name	Description	Type
teamlabid	ID	Text (string)
teamlabprojectownerid	ProjectOwner ID	Text (string)
teamlabtitle	Title	Text (string)
teamlaburl	URL	Text (string)

Trac

You can import tasks from your Trac instance using the `trac` service name.

Additional Dependencies

Install packages needed for Trac support with:

```
pip install bugwarrior[trac]
```

Example Service

Here's an example of a Trac target:

```
[my_issue_tracker]
service = trac
trac.base_uri = fedorahosted.org/moksha
trac.scheme = https
trac.project_template = moksha.{{traccomponent|lower}}
```

By default, this service uses the XML-RPC Trac plugin, which must be installed on the Trac instance. If this is not available, the service can use Trac's built-in CSV support, but in this mode it cannot add annotations based on ticket comments. To enable this mode, add `trac.no_xmlrpc = true`.

If your trac instance requires authentication to perform the query, add:

```
trac.username = ralph
trac.password = OMG_LULZ
```

The above example is the minimum required to import issues from Trac. You can also feel free to use any of the configuration options described in *Common Service Configuration Options*.

Service Features

Provided UDA Fields

Field Name	Description	Type
`tracnumber`	Number	Text (string)
`tracsummary`	Summary	Text (string)
`tracurl`	URL	Text (string)
`traccomponent`	Component	Text (string)

Trello

You can import tasks from Trello cards using the `trello` service name.

Options

`trello.api_key`
Your Trello API key, available from https://trello.com/app-key

`trello.token`
Trello token, see below for how to get it.

`trello.include_boards`
The list of board to include. If omitted, bugwarrior will use all boards the authenticated user is a member of. This can be either the board ids of the board "short links". The latter is the easiest option as it is part of the board URL: in your browser, navigate to the board you want to pull cards from and look at the URL, it should be something like `https://trello.com/b/xxxxxxxx/myboard`: copy the part between `/b/` and the next `/` in the `trello.include_boards` field.

trello.include_lists
If set, only pull cards from lists whose name is present in `trello.include_lists`.

trello.exclude_lists
If set, skip cards from lists whose name is present in `trello.exclude_lists`.

trello.import_labels_as_tags
A boolean that indicates whether the Trello labels should be imported as tags in taskwarrior. (Defaults to false.)

trello.label_template
Template used to convert Trello labels to taskwarrior tags. See *Field Templates* for more details regarding how templates are processed. The default value is `{{label|replace(' ', '_')}}`.

Example Service

Here's an example of a Trello target:

```
[my_project]
service = trello
trello.api_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
trello.token = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

The above example is the minimum required to import tasks from Trello. This will import every card from all the user's boards.

Here's an example with more options:

```
[my_project]
service = trello
trello.api_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
trello.token = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
trello.include_boards = AaBbCcDd, WwXxYyZz
trello.include_lists = Todo, Doing
trello.exclude_lists = Done
trello.only_if_assigned = someuser
trello.import_labels_as_tags = true
```

In this case, `bugwarrior` will only import cards from the specified boards if they belong to the right lists..

Feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

Service Features

Include and Exclude Certain Lists

You may want to pull cards from only a subset of the open lists in your board. To do that, you can use the `trello.include_lists` and `trello.exclude_lists` options.

For example, if you would like to only pull-in cards from your `Todo` and `Doing` lists, you could add this line to your service configuration:

```
trello.include_lists = Todo, Doing
```

Import Labels as Tags

Trello allows you to attach labels to cards; to use those labels as tags, you can use the `trello.import_labels_as_tags` option:

```
trello.import_labels_as_tags = True
```

Also, if you would like to control how these labels are created, you can specify a template used for converting the trello label into a Taskwarrior tag.

For example, to prefix all incoming labels with the string '*trello*' (perhaps to differentiate them from any existing tags you might have), you could add the following configuration option:

```
trello.label_template = trello_{{label}}
```

In addition to the context variable `{{label}}`, you also have access to all fields on the Taskwarrior task if needed.

Note: See *Field Templates* for more details regarding how templates are processed. The default value is `{{label|upper|replace(' ', '_')}}`.

Provided UDA Fields

Field Name	Description	Type
`trelloboard`	Board name	Text (string)
`trellocard`	Card name	Text (string)
`trellocardid`	Card ID	Text (string)
`trellolist`	List name	Text (string)
`trelloshortlink`	Short Link	Text (string)
`trelloshorturl`	Short URL	Text (string)
`trellourl`	Full URL	Text (string)

VersionOne

You can import tasks from VersionOne using the `versionone` service name.

Additional Requirements

Install the following package using `pip`:

- `v1pysdk-unofficial`

Example Service

Here's an example of a VersionOne project:

```
[my_issue_tracker]
service = versionone
versionone.base_uri = https://www3.v1host.com/MyVersionOneInstance/
versionone.usermame = somebody
versionone.password = hunter5
```

The above example is the minimum required to import issues from VersionOne. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

Note: This plugin does not infer a project name from any attribute of the version one Task or Story; it is recommended that you set the project name to use for imported tasks by either using the below *Set a Global Project Name* feature, or, if you require more flexibility, setting the `project_template` configuration option (see *Field Templates*).

Service Features

Restrict Task Imports to a Specific Timebox (Sprint)

You can restrict imported tasks to a specific Timebox (VersionOne's internal generic name for a Sprint) – in this example named 'Sprint 2014-09-22' – by using the `versionone.timebox_name` option; for example:

```
versionone.timebox_name = Sprint 2014-09-22
```

Set a Global Project Name

By default, this importer does not set a project name on imported tasks. Although you can gain more flexibility by using *Field Templates* to generate a project name, if all you need is to set a predictable project name, you can use the `versionone.project_name` option; in this example, to add imported tasks to the project 'important_project':

```
versionone.project_name = important_project
```

Set the Timezone Used for Due Dates

You can configure the timezone used for setting your tasks' due dates by setting the `versionone.timezone` option. By default, your local timezone will be used. For example:

```
versionone.timezone = America/Los_Angeles
```

Provided UDA Fields

Field Name	Description	Type
versiononetaskname	Task Name	Text (string)
versiononetaskoid	Task Object ID	Text (string)
versiononestoryoid	Story Object ID	Text (string)
versiononestoryname	Story Name	Text (string)
versiononetaskreference	Task Reference	Text (string)
versiononetaskdetailestimate	Task Detail Estimate	Text (string)
versiononetaskestimate	Task Estimate	Text (string)
versiononetaskdescrption	Task Description	Text (string)
versiononetasktodo	Task To Do	Text (string)
versiononestorydetailestimate	Story Detail Estimate	Text (string)
versiononestoryurl	Story URL	Text (string)
versiononetaskurl	Task URL	Text (string)
versiononestoryestimate	Story Estimate	Text (string)
versiononestorynumber	Story Number	Text (string)
versiononestorydescription	Story Description	Text (string)

YouTrack

You can import tasks from your YouTrack instance using the `youtrack` service name.

Example Service

Here's an example of a YouTrack target:

```
[my_issue_tracker]
service = youtrack
youtrack.host = youtrack.example.com
youtrack.login = turing
youtrack.password = 3n1Gm@
```

The above example is the minimum required to import issues from YouTrack. You can also feel free to use any of the configuration options described in *Common Service Configuration Options* or described in *Service Features* below.

Service Features

Customize the YouTrack Connection

The `youtrack.host` field is used to construct a URL for the YouTrack server. It defaults to a secure connection scheme (HTTPS) on the standard port (443).

To connect on a different port, set:

```
youtrack.port = 8443
```

If your YouTrack instance is only available over HTTP, set:

```
youtrack.use_https = False
```

If you want to ignore verifying the SSL certificate, set:

```
youtrack.verify_ssl = False
```

Specify the Query to Use for Gathering Issues

The default option selects unresolved issues assigned to the login user:

```
youtrack.query = for:me #Unresolved
```

Reference the YouTrack Search Query Grammar for additional examples.

Queries are capped at 100 max results by default, but may be adjusted to meet your needs:

```
youtrack.query_limit = 100
```

Import Issue Tags

The YouTrack issue tracker allows you to tag issues. To apply these tags in Taskwarrior, set:

```
youtrack.import_tags = True
```

If you would like to control how these tags are formatted, you can specify a template used for converting the YouTrack tag into a Taskwarrior tag.

For example, to prefix all incoming tags with the string 'yt_' (perhaps to differentiate them from any existing tags you might have), you could add the following configuration option:

```
youtrack.tag_template = yt_{{tag|lower}}
```

In addition to the context variable `{{tag}}`, you also have access to all fields on the Taskwarrior task if needed.

Note: See *Field Templates* for more details regarding how templates are processed.

Provided UDA Fields

Field Name	Description	Type
`youtrackissue`	PROJECT-ISSUE#	Text (string)
`youtracksummary`	Summary	Text (string)
`youtrackurl`	URL	Text (string)
`youtrackproject`	Project short name	Text (string)
`youtracknumber`	Project issue number	Numeric

Example Configuration

```
# Example bugwarriorrc

# General stuff.
[general]
# Here you define a comma separated list of targets.  Each of them must have a
# section below determining their properties, how to query them, etc.  The name
```

```
# is just a symbol, and doesn't have any functional importance.
targets = my_github, my_bitbucket, paj_bitbucket, moksha_trac, bz.redhat

# If unspecified, the default taskwarrior config will be used.
#taskrc = /path/to/.taskrc

# Setting this to true will shorten links with http://da.gd/
#shorten = False

# Setting this to True will include a link to the ticket in the description
inline_links = False

# Setting this to True will include a link to the ticket as an annotation
annotation_links = True

# Setting this to True will include issue comments and author name in task
# annotations
annotation_comments = True

# Defines whether or not issues should be matched based upon their description.
# In legacy mode, we will attempt to match issues to bugs based upon the
# presence of the '(bw)' marker in the task description.
# If this is false, we will only select issues having the appropriate UDA
# fields defined (which is smarter, better, newer, etc..)
legacy_matching = False

# log.level specifies the verbosity.  The default is DEBUG.
# log.level can be one of DEBUG, INFO, WARNING, ERROR, CRITICAL, DISABLED
#log.level = DEBUG

# If log.file is specified, output will be redirected there.  If it remains
# unspecified, output is sent to sys.stderr
#log.file = /var/log/bugwarrior.log

# Configure the default description or annotation length.
#annotation_length = 45

# Use hooks to run commands prior to importing from bugwarrior-pull.
# bugwarrior-pull will run the commands in the order that they are specified
# below.
#
# pre_import: The pre_import hook is invoked after all issues have been pulled
# from remote sources, but before they are synced to the TW db. If your
# pre_import script has a non-zero exit code, the `bugwarrior-pull` command will
# exit early.
[hooks]
pre_import = /home/someuser/backup.sh, /home/someuser/sometask.sh

# This section is for configuring notifications when bugwarrior-pull runs,
# and when issues are created, updated, or deleted by bugwarrior-pull.
# Three backends are currently supported:
#
#  - growlnotify (v2)   Mac OS X   "gntp" must be installed
#  - gobject            Linux      python gobject must be installed
#
# To configure, adjust the settings below.  Note that neither of the #
# "sticky" options have any effect on Linux.  They only work for
# growlnotify.
```

```
#[notifications]
# notifications = True
# backend = growlnotify
# finished_querying_sticky = False
# task_crud_sticky = True
# only_on_new_tasks = True


# This is a github example.  It says, "scrape every issue from every repository
# on http://github.com/ralphbean.  It doesn't matter if ralphbean owns the issue
# or not."
[my_github]
service = github
github.default_priority = H
github.add_tags = open_source

# This specifies that we should pull issues from repositories belonging
# to the 'ralphbean' github account.  See the note below about
# 'github.username' and 'github.login'.  They are different, and you need
# both.
github.username = ralphbean

# I want taskwarrior to include issues from all my repos, except these
# two because they're spammy or something.
github.exclude_repos = project_bar,project_baz

# Working with a large number of projects, instead of excluding most of them I
# can also simply include just a limited set.
github.include_repos = project_foo,project_foz

# Note that login and username can be different:  I can login as me, but
# scrape issues from an organization's repos.
#
# - 'github.login' is the username you ask bugwarrior to
#   login as.  Set it to your account.
# - 'github.username' is the github entity you want to pull
#   issues for.  It could be you, or some other user entirely.
github.login = ralphbean
github.password = OMG_LULZ


# Here's an example of a trac target.
[moksha_trac]
service = trac

trac.base_uri = fedorahosted.org/moksha
trac.username = ralph
trac.password = OMG_LULZ

trac.only_if_assigned = ralph
trac.also_unassigned = True
trac.default_priority = H
trac.add_tags = work

# Here's an example of a megaplan target.
[my_megaplan]
service = megaplan
```

```
megaplan.hostname = example.megaplan.ru
megaplan.login = alice
megaplan.password = secret
megaplan.project_name = example

# Here's an example of a jira project. The ``jira-python`` module is
# a bit particular, and jira deployments, like Bugzilla, tend to be
# reasonably customized. So YMMV. The ``base_uri`` must not have a
# have a trailing slash. In this case we fetch comments and
# cases from jira assigned to 'ralph' where the status is not closed or
# resolved.
[jira_project]
service = jira
jira.base_uri = https://jira.example.org
jira.username = ralph
jira.password = OMG_LULZ
jira.query = assignee = ralph and status != closed and status != resolved
# Set this to your jira major version. We currently support only jira version
# 4 and 5(the default). You can find your particular version in the footer at
# the dashboard.
jira.version = 5
jira.add_tags = enterprisey work

# Here's an example of a phabricator target
[my_phabricator]
service = phabricator
# No need to specify credentials.  They are gathered from ~/.arcrc

# Here's an example of a teamlab target.
[my_teamlab]
service = teamlab

teamlab.hostname = teamlab.example.com
teamlab.login = alice
teamlab.password = secret
teamlab.project_name = example_teamlab

# Here's an example of a redmine target.
[my_redmine]
service = redmine
redmine.url = http://redmine.example.org/
redmine.key = c0c4c014cafebabe
redmine.user_id = 7
redmine.project_name = redmine
redmine.add_tags = chiliproject

[activecollab]
service = activecollab
activecollab.url = https://ac.example.org/api.php
activecollab.key = your-api-key
activecollab.user_id = 15
activecollab.add_tags = php

[activecollab2]
service = activecollab2
activecollab2.url = http://ac.example.org/api.php
activecollab2.key = your-api-key
activecollab2.user_id = 15
```

```
activecollab2.projects = 1:first_project, 5:another_project

[my_gmail]
service = gmail
gmail.query = label:action OR label:readme
gmail.login_name = you@example.com
```

How to Contribute

Setting up your development environment

First, make sure you have the necessary *Requirements*.

You should also install the virtualenv tool for python. (I use a wrapper for it called virtualenvwrapper which is awesome but not required.) Virtualenv will help isolate your dependencies from the rest of your system.

```
$ sudo yum install python-virtualenv git
$ mkdir -p ~/virtualenvs/
$ virtualenv ~/virtualenvs/bugwarrior
```

You should now have a virtualenv in a `~/virtualenvs/` directory. To use it, you need to "activate" it like this:

```
$ source ~/virtualenv/bugwarrior/bin/activate
(bugwarrior)$ which python
```

At any time, you can deactivate it by typing `deactivate` at the command prompt.

Next step – get the code!

```
(bugwarrior)$ git clone git@github.com:ralphbean/bugwarrior.git
(bugwarrior)$ cd bugwarrior
(bugwarrior)$ python setup.py develop
(bugwarrior)$ which bugwarrior-pull
```

This will actually run it.. be careful and back up your task directory!

```
(bugwarrior)$ bugwarrior-pull
```

Making a pull request

Create a new branch for each pull request based off the `develop` branch:

```
(bugwarrior)$ git checkout -b my-shiny-new-feature develop
```

Please add tests when appropriate and run the test suite before opening a PR:

```
(bugwarrior)$ python setup.py nosetests
```

We look forward to your contribution!

Works in progress

The best way to get help and feedback before you pour too much time and effort into your branch is to open a "work in progress" pull request. We will not leave it open indefinitely if it doesn't seem to be progressing, but there's nothing to lose in soliciting some pointers and concerns.

Please begin the title of your work in progress pr with "[WIP]" and explain what remains to be done or what you're having trouble with.

FAQ

Can bugwarrior support <some issue tracking system>?

Sure! But our general rule here is that we won't write a backend for a service unless we use it personally, otherwise it's hard to be sure that it really works.

We also try to rely on people to become maintainers of the different backend plugins they use so that they don't suffer bit rot over time.

In summary, we need someone who 1) uses <some issue tracking system> and 2) can develop the plugin. Could it be you? :)

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index