Taskwarrior and Taskserver use a lot of terminology that is not obvious. Those terms are defined here.

active
When a task is started, like this:

```
$ task 1 start
```

It gains a `start` attribute, which is a date. When started, a task is considered *active*, until completed, deleted or stopped.

alias
An alias is name that can be used on the command line, which is replaced by its definition at run time. For example, you can use an alias to create a synonym for a command, like this:

```
$ task config alias.rm delete
```

Then whenever the `rm` alias is encountered, it is replaced by `delete`, making these two commands equivalent:

```
$ task 1 delete
$ task 2 rm
```

An alias can appear elsewhere on the command line, and is not limited to command synonyms.

annotation
An `annotation` is a note added to an existing task, and is in the form of a text string. There can be any number of annotations to a task.

```
$ task add Buy the milk
$ task 1 annotate and Bread
$ task 1 annotate and Cheese
```

Some reports show the annotations in full, some show an annotation count, and some simply show an indicator that there are annotations.

attribute
A task is comprised of a set of attrbitutes, which are simply name/value pairs. As an example, `project` is an attribute, the name being `project`, and the value being what you assign to that project.

```
$ task add project:Home Rake the leaves
```

Here you see the `project` attribute being directly set to `Home`, and the `description` attribute indirectly set to `Rake the leaves`.

calc 2.4.0
Taskwarrior includes a small utility program named `calc` that uses the Taskwarrior expression engine to calculate values. This utility mirrors the `calc` command. Here is an example:

```
$ calc '1 + 2 * 3'
7
```

The `calc` utility is mostly for testing, and allows access to internal features, such as postfix expressions:

```
$ calc --postfix '1 2 3 * +'
7
```

While this is just a simple example, there is also support for dates, durations, Boolean operations and many operators.

See Calc Command for full details.

certificate
A certificate is a document that identifies an entity. A certificate is an X.509 PEM file, part of PKI.

command
A Taskwarrior command can be a read-only command, wherein data is not modified:

```
list, info, calendar ...
```

Or it can be a write command, where data is modified:

```
add, modify, done, delete ...
```

The distinction is important because it dictates how the command line arguments are interpreted.

completed
`Completed` is one of several states a task can be in. The others are `pending`, `waiting`, `deleted` and `recurring`.

The `completed` state indicates the successful completion of a task..

custom report
Taskwarrior has several built-in reports, some of which are modifiable via configuration.

Additionally, you can create your own Custom Reports, where you can define the command, attributes show, length, filter, sorting and breaks.

The Report gives a detailed explanation of how to use, modify and create reports.

daemon
A daemon is the name given to a process that runs without interaction as a background process. Various server products are run as daemons, and are automatically launched on startup. Typically daemons are named ending with 'd'.

Taskserver (taskd) can be run as a daemon, using the `--daemon` command line argument.

dateformat
TBD

default command
Taskwarrior supports the notion of a default command, which is used when no other command is specified on the command line. Suppose you have this setting:

```
$ task config default.command list
$ task +tag
```

Then the default command above of `list` would be used in the subsequent command, to effectively become:

```
$ task +tag list
```

2.4.0 When no command is specified, and the only arguments are task `ID` or `UUID` values, the `info` command is assumed, regardless of the `rc.default.command` setting.

deleted
`Deleted` is one of several states a task can be in. The others are `pending`, `waiting`, `completed` and `recurring`.

The `deleted` state indicates an incomplete task that is removed from the list, perhaps because it is no longer relevant, or someone else completed it.

dogfood
Refers to the phrase "eating one's own dogfood". This describes the practice of using a product while developing that product. This encourages improvements in the product right from the start.

dependency
TBD

due
A due date may be applied to a task, like this:

```
$ task 1 modify due:25th
```

A task with a due date has a higher urgency value, and generally sorts higher in reports. The due date is considered the date on which the task must be completed.

A task is consided due, if the due date is within 7 days in the future. The '7' is configurable via the `due` configuration setting. If a due date is further into the future it is not considered due. If the due date is in the past, it is considered overdue, and these tasks are shown in the `overdue` report.

end
When a task is completed or deleted, it is given an `end` date.

entry
When a task is first created, it is automatically assigned an `entry` date. This attribute is used to calculate the age of a task. This date is modifiable, should you wish to enter an old task.

en-passant
En-passant is used to describe the action of annotating or otherwise modifying a task while making some other change to it. An example is deleting a task, and adding an explanatory note at the same time. Instead of doing this:

```
$ task 123 annotate Could not reproduce bug
$ task 123 modify +cannot_reproduce
$ task 123 delete
```

Those three actions can be combined:

```
$ task 123 delete Could not reproduce bug +cannot_reproduce
```

For every write command, except `modify`, you can perform *en-passant* modifications. The `modify` is different, and updates the task description instead of annotating.

export
Task can be exported from Taskwarrior, using the `export` command. You can apply a filter to the export command to control which tasks are exported.

Tasks are exported using the Taskwarrior JSON Format.

```
$ task add Buy milk
$ task /milk/ export
{"description":"Buy milk","entry":"20140928T211124Z","status":"pending","uuid":"2e17b750-c137-4ddd-b6dd-c63e272107f4"}
```

Exported tasks may be imported. This is the recommended mechanism for writing add-on scripts and programs.

expression 2.4.0
An algebraic expression is a mathematical expression comprised of constants, operators and symbols. Here are some examples of expressions in the context of a Taskwarrior command line:

```
1
1 + 2
1 2 +
123.due + 4 weeks
pi * r^2
```

Expressions of this kind can appear in filters or modifications.

extension 2.4.0
An extension is a program that adds functionality to Taskwarrior. This could be in the form of a wrapper (Vit), or an aliased script, or a hook script.

filter
A filter is a set of command line arguments used to restrict the number of tasks shown. For example, this command uses three filter terms:

```
$ task project:Home +kitchen +weekend list
```

This command will show a report that contains tasks that have the project value 'Home', and have the `kitchen` and `weekend` tags. The three terms are combined with an implicit `AND` operator.

In addition, the command above combines the three terms in the filter with the implicit filter of the `list` command, which is:

```
status:pending
```

So in total, the command has four filter terms, that restrict the tasks shown. A filter is essentially an ælgebraic expression that must evalute to `True` for each task, for it to be included in the result.

Most commands accept filters. See Filters for a more complete discussion.

gc

Taskwarrior performs a Garbage Collect (gc) operation whenever it is necessary. Tasks that are deleted and completed are moved from the `pending.data` file to the `completed.data` file. This process keeps the `pending.data` file compact and fast to read, and keeps the ID numbers as low as possible.

Although gc can be disabled, to keep the ID numbers static, this is considered a bad idea.

holiday file

TBD

hooks 2.4.0

Hooks are a mechanism that allows other programs/scripts to run at certain points in Taskwarrior's execution, so that processing can be influenced. This is an extension mechanism.

See the Hooks Design document.

import

TBD

Infix

Infix is a mathematical notation for expressions where the operator appears between its operands. This example:

```
1 + 2
```

Represents the sum of two digits, with the `+` operator in the infix position, between the numbers.

info

The `info` report (actual full name `information`) is a report that shows *all* task metadata in a human-readble form. Additionally, when a task ID (or UUID) is provided, and no command is specified, the `info` command is assumed:

```
$ task 123
```

...

ISO-8601

TBD

JSON

JSON (JavaScript Object Notation) is an industry-standard, lightweight, data-interchange format. It is easy for humans and machines to read and write. It is well supported by most languages.

JSON is the only import/export format supported by Taskwarrior. Support for any other format requires conversion between JSON and the other format. There are several examples of this provided with Taskwarrior, in the `scripts/add-ons` directory.

See www.json.org for a full definition.

key

A key is a document that specifies an encryption key. A key is an X.509 PEM file, part of PKI.

list break 2.4.0
A list break is associated with report sorting, such that when a report is sorted on an attribute, a list break for that attribute causes a blank line to be inserted between unique values of that attribute. For example, the sort definition:

```
$ task show report.list.sort

Config Variable  Value
---------------- ------------------------------------
report.list.sort start-,due+,project+/,urgency-
```

Here the `project` attribute is sorted ascending and also has a list break, indicated by `/`.

override
An override refers to the action of superceding a configuration setting at runtime. For example, support for `rc.color` is 'on' by default, but at runtime, you can override this:

```
$ task rc.color=off list
```

This runs the `list` report but does not use color.

Any number of settings can be overridden at runtime, and overrides can also appear in the `rc.report.<report>.filter` setting for a report, so that you can have a report run without color, for example.

pending
`Pending` is one of several states a task can be in. The others are `deleted`, `waiting`, `completed` and `recurring`.

The `pending` state indicates an incomplete task that is yet to be completed or deleted.

PKI
PKI (Public Key Infrastructure) is the set of mechanisms and policies built around certificates, including their creation, validation and handling.

See PKI

rc
TBD

ready
Ready is the name of a report that shows ready tasks. A task is considered *ready* if it is pending, is not blocked, and either has no scheduled date, or has one that is before now.

regex
A regex, or **Reg**ular **Ex**pression is a very powerful mechanism for searching and matching text. It is a complex language that describes search patterns, and allows for sophisticated searching.

Suppose you were solving a crossword puzzle, and needed a five letter word that started and ended in a vowel, you could represent that in regex, and it could look something like:

```
[aeiou]...[aeiou]
```

See Wikipedia: Regular expression.

Report
TBD

RPN
Reverse **P**olish **N**otation, or postfix notation is a mathematical notation. This example:

```
1 2 +
```

Represents the sum of two digits, with the `+` operator in the postfix position, last.

Writing software to evaluate postfix expressions is significantly easier than for infix notation, which is easier for humans to read.

scheduled
The `scheduled` date represents the first opportunity to begin work on a task, and therefore typically is set to be on or after the wait date, but before the due date.

If you add a scheduled date to a task, like this:

```
$ task 1 modify scheduled:2014-12-31
```

Then this task will be considered *ready* after that date. The virtual tag `READY` reflects this, and the report `ready` shows such tasks.

shell
The shell is your command interpreter. Most likely you are using the `bash` shell, but there are many others.

Any program that has complex argument processing will likely face several problems that are shell-related. The shell modifies the entered text before passing it on to the program. For example, consider these two commands:

```
$ ls my_file
$ ls 'my_file'
```

In both cases, the `ls` command sees the same command line argument, `my_file`, because the shell removes the quotes. There are many other cases to be aware of also.

shadow file
The shadow file is an old feature of Taskwarrior, which automatically ran a report whenever the data changed. That report was configurable, and the output placed in a text file. This was typically used to display a task list on the desktop using Conky, or similar tools.

Starting with version 2.4.0, shadow file support has been removed, in favor of a hook-based solution, an example of which is provided with the source.

special tag
A special tag is one that, when applied to a task, changes the behavior of Taskwarrior for that task. There are several special tags supported:

+nocolor	Disable color rules processing for this task
+nonag	Completion of this task suppresses all nag messages
+nocal	This task will not appear on the calendar
+next	Elevates task so it appears on 'next' report

See Tag, Virtual Tags.

start
If you apply a `start` date to a task, then the task is considered *active*. Active tasks have a higher urgency value, and can be listed using the `active` command. There are two ways to apply a start date, the latter giving you control over the recorded start date:

```
$ task 1 start
$ task 1 modify start:now
```

If the configuration setting `journal.info` is on, then when a task is started or stopped, an annotation is made to that effect. The cumulative start/stop times can be viewed in the info report.

Note that this does not imply there is time tracking in Taskwarrior, this is simply a log of start/stop times, and is not modifiable. Many users confuse this log with legitimate time tracking capabilities. Don't make this mistake.

substitution
Taskwarrior can make command line substitutions like this:

```
$ task 123 modify /this/that/
```

The substitution syntax allows text replacement in task description and annotations.

sync
Synchronization is a service provided by the Taskserver, and allows multiple sync clients to share data. The Taskserver knows how to merge tasks and handle incremental updates to minimize bandwidth used.

Although you can achieve a similar capability using one of the many file-sharing services, the merging that happens is not aware of how to merge tasks.

Starting with Taskwarrior 2.3.0, there is a `sync` command that can be configured to talk to a Taskserver.

tag
A task may have any number of tags attached to it. A tag is a single word (no spaces), and can be included when the task is created:

```
$ task add Paint the walls +home +weekend
```

This adds two tags `home` and `weekend` to the task. You can modify a task to add a `renovate` tag:

```
$ task 1 modify +renovate
```

Or remove a `weekend` tag:

```
$ task 1 modify -weekend
```

See Special Tags, Virtual Tags.

theme
A taskwarrior theme is a file containing color definitions. There are several themes provided with Taskwarrior, and these are listed in your `.taskrc` file, and if you uncomment one of these lines, then the them is included, and you see a set of chosen and thematic colors. Your `.taskrc` file should contain this section:

```
# Color theme (uncomment one to use)
#include /usr/local/share/doc/task/rc/light-16.theme
#include /usr/local/share/doc/task/rc/light-256.theme
#include /usr/local/share/doc/task/rc/dark-16.theme
#include /usr/local/share/doc/task/rc/dark-256.theme
#include /usr/local/share/doc/task/rc/dark-red-256.theme
#include /usr/local/share/doc/task/rc/dark-green-256.theme
#include /usr/local/share/doc/task/rc/dark-blue-256.theme
#include /usr/local/share/doc/task/rc/dark-violets-256.theme
#include /usr/local/share/doc/task/rc/dark-yellow-green.theme
#include /usr/local/share/doc/task/rc/dark-gray-256.theme
#include /usr/local/share/doc/task/rc/dark-default-16.theme
#include /usr/local/share/doc/task/rc/dark-gray-blue-256.theme
```

All of these lines are said to be 'commented out', which means the '#' symbol at the beginning of the line prevents Taskwarrior from reading the rest of the line. Removing a '#' enables the color theme. For example, to enable the `dark-violets-256.theme` file, change the above to look like this:

```
# Color theme (uncomment one to use)
#include /usr/local/share/doc/task/rc/light-16.theme
#include /usr/local/share/doc/task/rc/light-256.theme
#include /usr/local/share/doc/task/rc/dark-16.theme
#include /usr/local/share/doc/task/rc/dark-256.theme
#include /usr/local/share/doc/task/rc/dark-red-256.theme
#include /usr/local/share/doc/task/rc/dark-green-256.theme
#include /usr/local/share/doc/task/rc/dark-blue-256.theme
include /usr/local/share/doc/task/rc/dark-violets-256.theme
#include /usr/local/share/doc/task/rc/dark-yellow-green.theme
#include /usr/local/share/doc/task/rc/dark-gray-256.theme
#include /usr/local/share/doc/task/rc/dark-default-16.theme
#include /usr/local/share/doc/task/rc/dark-gray-blue-256.theme
```

UDA

Taskwarrior supports a set of about 20 or so core attributes, which include ID, descriptions, tags, due date and so on. These attributes are necessary for the basic features provided.

But suppose you wanted to track a time estimate. There is no attribute for that, although you could store data in an attribute. A better solution is a User Defined Attribute, named `estimate`, of type `numeric`, that you define via configuration like this:

```
$ task config uda.estimate.label Estimate
$ task config uda.estimate.type duration
```

Once you have defined a UDA, you use it just like you would use any other core attribute, like this:

```
$ task add Paint the door due:eom estimate:4hours
```

Taskwarrior will faithfully store, report, sort and filter UDA attributes, but of course their true meaning is not understood, and so there are no features surrounding these values.

UDAs may contribute to the urgency calculation, using:

```
urgency.uda.<name>.coefficient
```

UDAs may have default values (`uda.<name>.default`), allowable values (`uda.<name>.values`), types (`string`, `date`, `duration`, and `numeric`) and finally a label for report headers (`uda.<name>.label`).

See User Defined Attributes (UDAs) for more details.

until
The `until` date represents the date that a task is automatically deleted. It is an expiration date. If you add an until date like this:

```
$ task 1 modify until:eoy
```

Then this task will expire and disappear at the end of the year.

urgency
The urgency of a task is a numeric quantity that represents task attributes that convey importance. Consider these two tasks:

```
$ task add Buy the paint   due:saturday +home +renovate +store
$ task add Paint the walls due:sunday   +home +renovate
```

The first task is due sooner, and there is no question that it is more important than the second, after all, the second is blocked by the first. Dependencies could be used here to represent this, but this is just an example.

We would like the first task to sort higher than the second task, in this case because of the due date. But not every task has (or needs) a due date. Urgency is therefore a way to raise and lower tasks in the sorted list. This cannot be achieved by simply providing a complex sort order, because usage patterns are different for everyone.

Urgency is implemented as a polynomial, where the term weights are configurable to make an attribute an important contributor to urgency (high positive number), a marginal contributor (low positive number), a detractor (negative number) or have no influence at all (zero).

While the urgency value in isolation is meaningless, when compared to that of other tasks, helps order the tasks by importance.

The `urgency` attribute is featured in the `next` report, although can be used in any report.

As the polynomial term weights (coefficients) are all modifiable, you can tweak them to make the urgency value more closely match your own notions of importance. Provided you understand the way the coefficients are used, this can be very beneficial.

See Urgency.

UUID
A **U**niversally **U**nique **ID**entifier is a 128-bit number used to uniquely identify a task. Whereas the ID of a task is simply a line number in the `pending.data` file, the UUID is permanent.

You can use ID and UUID interchangeably, although the ID is much more convenient.

```
$ task _get 1.uuid
8ad2e3db-914d-4832-b0e6-72fa04f6e331
$ task _get 8ad2e3db-914d-4832-b0e6-72fa04f6e331.id
1
```

A UUID never changes, and is not modifiable.

verbosity
Verbosity refers to the output Taskwarrior generates. For example if you run a report, the output might looks like this (data excluded for clarity):

```
$ task next rc.color=off

ID  Age  Project Tag Description              Urg
--- ---- ------- --- ------------------------ ----
  1 2wks    Home      Paint the walls          1.2
...

31 tasks
Configuration override rc.color:off
Filter: ( ( status == pending ) )
```

Aside from the actual data, there is an initial blank line, a set of column headers, another blank line, the summary '31 tasks', the configuration override, and the filter used. All of these decorative elements, and more, are controlled by the verbosity settings in the `verbose` configuration setting.

See Verbosity.

virtual tag
The tag notation for filters is convenient and simple, for example:

```
$ task +HOME list
...
$ task -HOME list
```

The first lists only tasks that have the `HOME` tag, and the second lists only tasks that do not have the `HOME` tag.

A virtual tag is a means of performing simple filtering, using tags that are synthesized at run time, and represent the complex internal state of a task. For example the virtual tag `ACTIVE` is an easy way to filter tasks that have been started, so that this command:

```
$ task +ACTIVE list
```

Is a more friendly and simple way to represent this equivalent command:

```
$ task start.any: list
```

See Tag, Special Tags.

wait
If you add a `wait` date to a task like this:

```
$ task 1 modify wait:30th
```

Then the task is hidden from reports until the 30th of the month, whereupon the wait date is removed, and the task becomes visible.

This is useful for tasks that are far into the future, where you may not appreciate seeing that task on every list, at least until it gets closer to the due date.

Note that a due date is not required to use this feature.

waiting
The `waiting` command is a report that shows tasks that are currently hidden because they have a future `wait` date.

`Waiting` is also one of several states a task can be in. The others are `pending`, `completed`, `deleted` and `recurring`.

The `waiting` state indicates a pending task that is hidden from view.

active
alias
annotation
attribute
calc
certificate
command
completed
custom report
daemon
dateformat
default command
deleted
dependency
dogfood
due
end
entry
en-passant
export
expression
extension
filter
gc
holiday file
hooks
import
Infix
info
ISO-8601
JSON
key
list break
override
pending
PKI
rc
ready
regex
Report
RPN
scheduled
shadow file
shell
special tag
start
substitution
sync
tag
theme
UDA
until
urgency
UUID
verbosity
virtual tag
wait
waiting