vdirsyncer Documentation

Release 0.16.3

Markus Unterwaditzer

Oct 03, 2017

Users

- Documentation
- Source code

Vdirsyncer is a command-line tool for synchronizing calendars and addressbooks between a variety of servers and the local filesystem. The most popular usecase is to synchronize a server with a local folder and use a set of other *programs* to change the local events and contacts. Vdirsyncer can then synchronize those changes back to the server.

However, vdirsyncer is not limited to synchronizing between clients and servers. It can also be used to synchronize calendars and/or addressbooks between two servers directly.

It aims to be for calendars and contacts what OfflineIMAP is for emails.

CHAPTER 1

When do I need Vdirsyncer?

Why not Dropbox + todo.txt?

Projects like todo.txt criticize the complexity of modern productivity apps, and that rightfully. So they set out to create a new, super-simple, human-readable format, such that vim suffices for viewing the raw data. However, when they're faced with the question how to synchronize that data across multiple devices, they seemed to have reached the dead end with their novel idea: "Let's just use Dropbox".

What does file sync software do if both files have changed since the last sync? The answer is to ignore the question, just sync as often as possible, and hope for the best. Because if it comes to a sync conflict, most sync services are not daring to merge files, and create two copies on each computer instead. Merging the two task lists is left to the user.

A better idea would've been to use `git` to synchronize the `todo.txt` file, which is at least able to resolve some basic conflicts.

Why not file sync (Dropbox, git, ...) + vdir?

Since *vdirs* are just a bunch of files, it is obvious to try *file synchronization* for synchronizing your data between multiple computers, such as:

- Syncthing
- Dropbox or one of the gajillion services like it
- unison
- Just `git` with a `sshd`.

The disadvantages of those solutions largely depend on the exact file sync program chosen:

- Like with `todo.txt`, Dropbox and friends are obviously agnostic/unaware of the files' contents. If a file has changed on both sides, Dropbox just copies both versions to both sides.

This is a good idea if the user is directly interfacing with the file system and is able to resolve conflicts themselves. Here it might lead to erroneous behavior with e.g. `khal`, since there are now two events with the same UID.

This point doesn't apply to git: It has very good merging capabilities, better than what vdirsyncer currently has.

- Such a setup doesn't work at all with smartphones. Vdirsyncer, on the other hand, synchronizes with CardDAV/CalDAV servers, which can be accessed with e.g. DAVDroid or the apps by dmfs.

CHAPTER 2

Installation

OS/distro packages

The following packages are user-contributed and were up-to-date at the time of writing:

- ArchLinux (AUR)
- Ubuntu and Debian, x86_64-only (packages also exist in the official repositories but may be out of date)
- GNU Guix
- OS X (homebrew)
- BSD (pkgsrc)
- OpenBSD

We only support the latest version of vdirsyncer, which is at the time of this writing 0.16.3. Please **do not file bugs if you use an older version**.

Some distributions have multiple release channels. Debian and Fedora for example have a "stable" release channel that ships an older version of vdirsyncer. Those versions aren't supported either.

If there is no suitable package for your distribution, you'll need to *install vdirsyncer manually*. There is an easy command to copy-and-paste for this as well, but you should be aware of its consequences.

Manual installation

If your distribution doesn't provide a package for vdirsyncer, you still can use Python's package manager "pip". First, you'll have to check that the following things are installed:

- Python 3.4+ and pip.
- `libxml` and `libxslt`
- `zlib`

- Linux or OS X. **Windows is not supported, see :gh:'535'.**

On Linux systems, using the distro's package manager is the best way to do this, for example, using Ubuntu:

```
sudo apt-get install libxml2 libxslt1.1 zlib1g python
```

Then you have several options. The following text applies for most Python software by the way.

The dirty, easy way

The easiest way to install vdirsyncer at this point would be to run:

```
pip install --user --ignore-installed vdirsyncer
```

- `--user` is to install without root rights (into your home directory)
- `--ignore-installed` is to work around Debian's potentially broken packages (see *Requests-related ImportErrors*).

This method has a major flaw though: Pip doesn't keep track of the files it installs. Vdirsyncer's files would be located somewhere in `~/.local/lib/python*`, but you can't possibly know which packages were installed as dependencies of vdirsyncer and which ones were not, should you decide to uninstall it. In other words, using pip that way would pollute your home directory.

The clean, hard way

There is a way to install Python software without scattering stuff across your filesystem: virtualenv. There are a lot of resources on how to use it, the simplest possible way would look something like:

```
virtualenv ~/vdirsyncer_env
~/vdirsyncer_env/bin/pip install vdirsyncer
alias vdirsyncer="~/vdirsyncer_env/bin/vdirsyncer
```

You'll have to put the last line into your `.bashrc` or `.bash_profile`.

This method has two advantages:

- It separately installs all Python packages into `~/vdirsyncer_env/`, without relying on the system packages. This works around OS- or distro-specific issues.
- You can delete `~/vdirsyncer_env/` to uninstall vdirsyncer entirely.

The clean, easy way

pipsi is a new package manager for Python-based software that automatically sets up a virtualenv for each program you install. Assuming you have it installed on your operating system, you can do:

```
pipsi install --python python3 vdirsyncer
```

and `.local/bin/vdirsyncer` will be your new vdirsyncer installation. To update vdirsyncer to the latest version:

```
pipsi upgrade vdirsyncer
```

If you're done with vdirsyncer, you can do:

```
pipsi uninstall vdirsyncer
```

and vdirsyncer will be uninstalled, including its dependencies.

CHAPTER 3

Tutorial

Before starting, *consider if you actually need vdirsyncer*. There are better alternatives available for particular usecases.

Installation

See *Installation*.

Configuration

Note:

- The config.example from the repository contains a very terse version of this.
- In this example we set up contacts synchronization, but calendar sync works almost the same. Just swap `type = "carddav"` for `type = "caldav"` and `fileext = ".vcf"` for `fileext = ".ics"`.
- Take a look at the *Known Problems* page if anything doesn't work like planned.

By default, vdirsyncer looks for its configuration file in the following locations:

- The file pointed to by the `VDIRSYNCER_CONFIG` environment variable.
- `~/.vdirsyncer/config`.
- `$XDG_CONFIG_HOME/vdirsyncer/config`, which is normally `~/.config/vdirsyncer/config`. See the XDG-Basedir specification.

The config file should start with a *general section*, where the only required parameter is `status_path`. The following is a minimal example:

```
[general]
status_path = "~/.vdirsyncer/status/"
```

After the general section, an arbitrary amount of *pair and storage sections* might come.

In vdirsyncer, synchronization is always done between two storages. Such storages are defined in *storage sections*, and which pairs of storages should actually be synchronized is defined in *pair section*. This format is copied from OfflineIMAP, where storages are called repositories and pairs are called accounts.

The following example synchronizes ownCloud's addressbooks to `~/.contacts/`:

```
[pair my_contacts]
a = "my_contacts_local"
b = "my_contacts_remote"
collections = ["from a", "from b"]

[storage my_contacts_local]
type = "filesystem"
path = "~/.contacts/"
fileext = ".vcf"

[storage my_contacts_remote]
type = "carddav"

# We can simplify this URL here as well. In theory it shouldn't matter.
url = "https://owncloud.example.com/remote.php/carddav/"
username = "bob"
password = "asdf"
```

Note: Configuration for other servers can be found at *Servers*.

After running `vdirsyncer discover` and `vdirsyncer sync`, `~/.contacts/` will contain subfolders for each addressbook, which in turn will contain a bunch of `.vcf` files which all contain a contact in `VCARD` format each. You can modify their contents, add new ones and delete some[1], and your changes will be synchronized to the CalDAV server after you run `vdirsyncer sync` again. For further reference, it uses the storages *`filesystem`* and *`carddav`*.

However, if new collections are created on the server, it will not automatically start synchronizing those[2]. You need to run `vdirsyncer discover` again to re-fetch this list instead.

More Configuration

Conflict resolution

What if the same item is changed on both sides? What should vdirsyncer do? Three options are currently provided:

1. vdirsyncer displays an error message (the default);
2. vdirsyncer chooses one alternative version over the other;
3. vdirsyncer starts a command of your choice that is supposed to merge the two alternative versions.

Options 2 and 3 require adding a `"conflict_resolution"` parameter to the pair section. Option 2 requires giving either `"a wins"` or `"b wins"` as value to the parameter:

[1] You'll want to *use a helper program for this*.
[2] Because collections are added rarely, and checking for this case before every synchronization isn't worth the overhead.

```
[pair my_contacts]
...
conflict_resolution = "b wins"
```

Earlier we wrote that `b = "my_contacts_remote"`, so when vdirsyncer encounters the situation where an item changed on both sides, it will simply overwrite the local item with the one from the server.

Option 3 requires specifying as value of `"conflict_resolution"` an array starting with `"command"` and containing paths and arguments to a command. For example:

```
[pair my_contacts]
...
conflict_resolution = ["command", "vimdiff"]
```

In this example, `vimdiff <a> <b>` will be called with `<a>` and `<b>` being two temporary files containing the conflicting files. The files need to be exactly the same when the command returns. More arguments can be passed to the command by adding more elements to the array.

See *Pair Section* for the reference documentation.

Metadata synchronization

Besides items, vdirsyncer can also synchronize metadata like the addressbook's or calendar's "human-friendly" name (internally called "displayname") or the color associated with a calendar. For the purpose of explaining this feature, let's switch to a different base example. This time we'll synchronize calendars:

```
[pair my_calendars]
a = "my_calendars_local"
b = "my_calendars_remote"
collections = ["from a", "from b"]
metadata = ["color"]

[storage my_calendars_local]
type = "filesystem"
path = "~/.calendars/"
fileext = ".ics"

[storage my_calendars_remote]
type = "caldav"

url = "https://owncloud.example.com/remote.php/caldav/"
username = "bob"
password = "asdf"
```

Run `vdirsyncer discover` for discovery. Then you can use `vdirsyncer metasync` to synchronize the `color` property between your local calendars in `~/.calendars/` and your ownCloud. Locally the color is just represented as a file called `color` within the calendar folder.

More information about collections

"Collection" is a collective term for addressbooks and calendars. Each collection from a storage has a "collection name", a unique identifier for each collection. In the case of *`filesystem`*-storage, this is the name of the directory that represents the collection, in the case of the DAV-storages this is the last segment of the URL. We use this identifier in the `collections` parameter in the `pair`-section.

This identifier doesn't change even if you rename your calendar in whatever UI you have, because that only changes the so-called "displayname" property[3]. On some servers (iCloud, Google) this identifier is randomly generated and has no correlation with the displayname you chose.

There are three collection names that have a special meaning:

- `"from a"`, `"from b"`: A placeholder for all collections that can be found on side A/B when running `vdirsyncer discover`.
- `null`: The parameters give to the storage are exact and require no discovery.

The last one requires a bit more explanation. Assume this config which synchronizes two directories of addressbooks:

```
[pair foobar]
a = "foo"
b = "bar"
collections = ["from a", "from b"]

[storage foo]
type = "filesystem"
fileext = ".vcf"
path = "./contacts_foo/"

[storage bar]
type = "filesystem"
fileext = ".vcf"
path = "./contacts_bar/"
```

As we saw previously this will synchronize all collections in `./contacts_foo/` with each same-named collection in `./contacts_bar/`. If there's a collection that exists on one side but not the other, vdirsyncer will ask whether to create that folder on the other side.

If we set `collections = null`, `./contacts_foo/` and `./contacts_bar/` are no longer treated as folders with collections, but as collections themselves. This means that `./contacts_foo/` and `./contacts_bar/` will contain `.vcf`-files, not subfolders that contain `.vcf`-files.

This is useful in situations where listing all collections fails because your DAV-server doesn't support it, for example. In this case, you can set `url` of your *carddav*- or *caldav*-storage to a URL that points to your CalDAV/CardDAV collection directly.

Note that not all storages support the `null`-collection, for example *google_contacts* and *google_calendar* don't.

Advanced collection configuration (server-to-server sync)

The examples above are good enough if you want to synchronize a remote server to a previously empty disk. However, even more trickery is required when you have two servers with *already existing* collections which you want to synchronize.

The core problem in this situation is that vdirsyncer pairs collections by collection name by default (see definition in previous section, basically a foldername or a remote UUID). When you have two servers, those collection names may not line up as nicely. Suppose you created two calendars "Test", one on a NextCloud server and one on iCloud, using their respective web interfaces. The URLs look something like this:

```
NextCloud: https://example.com/remote.php/dav/calendars/user/test/
iCloud:    https://p-XX.caldav.icloud.com/YYY/calendars/3b4c9995-5c67-4021-9fa0-
↪be4633623e1c
```

[3] Which you can also synchronize with `metasync` using `metadata = ["displayname"]`.

Those are two DAV calendar collections. Their collection names will be `test` and `3b4c9995-5c67-4021-9fa0-be4633623e1c` respectively, so you don't have a single name you can address them both with. You will need to manually "pair" (no pun intended) those collections up like this:

```
[pair doublecloud]
a = "my_nextcloud"
b = "my_icloud"
collections = [["mytest", "test", "3b4c9995-5c67-4021-9fa0-be4633623e1c"]]
```

`mytest` gives that combination of calendars a nice name you can use when talking about it, so you would use `vdirsyncer sync doublecloud/mytest` to say: "Only synchronize these two storages, nothing else that may be configured".

Note: Why not use displaynames?

You may wonder why vdirsyncer just couldn't figure this out by itself. After all, you did name both collections "Test" (which is called "the displayname"), so why not pair collections by that value?

There are a few problems with this idea:

- Two calendars may have the same exact displayname.
- A calendar may not have a (non-empty) displayname.
- The displayname might change. Either you rename the calendar, or the calendar renames itself because you change a language setting.

In the end, that property was never designed to be parsed by machines.

CHAPTER 4

SSL and certificate validation

All SSL configuration is done per-storage.

Pinning by fingerprint

To pin the certificate by fingerprint:

```
[storage foo]
type = "caldav"
...
verify_fingerprint = "94:FD:7A:CB:50:75:A4:69:82:0A:F8:23:DF:07:FC:69:3E:CD:90:CA"
#verify = false  # Optional: Disable CA validation, useful for self-signed certs
```

SHA1-, SHA256- or MD5-Fingerprints can be used. They're detected by their length.

You can use the following command for obtaining a SHA-1 fingerprint:

```
echo -n | openssl s_client -connect unterwaditzer.net:443 | openssl x509 -noout -
↪fingerprint
```

Note that `verify_fingerprint` doesn't suffice for vdirsyncer to work with self-signed certificates (or certificates that are not in your trust store). You most likely need to set `verify = false` as well. This disables verification of the SSL certificate's expiration time and the existence of it in your trust store, all that's verified now is the fingerprint.

However, please consider using Let's Encrypt such that you can forget about all of that. It is easier to deploy a free certificate from them than configuring all of your clients to accept the self-signed certificate.

Custom root CAs

To point vdirsyncer to a custom set of root CAs:

```
[storage foo]
type = "caldav"
...
verify = "/path/to/cert.pem"
```

Vdirsyncer uses the requests library, which, by default, uses its own set of trusted CAs.

However, the actual behavior depends on how you have installed it. Many Linux distributions patch their `python-requests` package to use the system certificate CAs. Normally these two stores are similar enough for you to not care.

But there are cases where certificate validation fails even though you can access the server fine through e.g. your browser. This usually indicates that your installation of the `requests` library is somehow broken. In such cases, it makes sense to explicitly set `verify` or `verify_fingerprint` as shown above.

Client Certificates

Client certificates may be specified with the `auth_cert` parameter. If the key and certificate are stored in the same file, it may be a string:

```
[storage foo]
type = "caldav"
...
auth_cert = "/path/to/certificate.pem"
```

If the key and certificate are separate, a list may be used:

```
[storage foo]
type = "caldav"
...
auth_cert = ["/path/to/certificate.crt", "/path/to/key.key"]
```

CHAPTER 5

Storing passwords

Changed in version 0.7.0: Password configuration got completely overhauled.

Vdirsyncer can fetch passwords from several sources other than the config file.

Command

Say you have the following configuration:

```
[storage foo]
type = "caldav"
url = ...
username = "foo"
password = "bar"
```

But it bugs you that the password is stored in cleartext in the config file. You can do this:

```
[storage foo]
type = "caldav"
url = ...
username = "foo"
password.fetch = ["command", "~/get-password.sh", "more", "args"]
```

You can fetch the username as well:

```
[storage foo]
type = "caldav"
url = ...
username.fetch = ["command", "~/get-username.sh"]
password.fetch = ["command", "~/get-password.sh"]
```

Or really any kind of parameter in a storage section.

With pass for example, you might find yourself writing something like this in your configuration file:

```
password.fetch = ["command", "pass", "caldav"]
```

Accessing the system keyring

As shown above, you can use the `command` strategy to fetch your credentials from arbitrary sources. A very common usecase is to fetch your password from the system keyring.

The keyring Python package contains a command-line utility for fetching passwords from the OS's password store. Installation:

```
pip install keyring
```

Basic usage:

```
password.fetch = ["command", "keyring", "get", "example.com", "foouser"]
```

Password Prompt

You can also simply prompt for the password:

```
[storage foo]
type = "caldav"
username = "myusername"
password.fetch = ["prompt", "Password for CalDAV"]
```

CHAPTER 6

Syncing with read-only storages

If you want to subscribe to a public, read-only WebCAL-calendar but neither your server nor your calendar apps support that (or support it insufficiently), vdirsyncer can be used to synchronize such a public calendar `A` with a new calendar `B` of your own and keep `B` updated.

Step 1: Create the target calendar

First you need to create the calendar you want to sync the WebCAL-calendar with. Most servers offer a web interface for this. You then need to note the CalDAV URL of your calendar. Note that this URL should directly point to the calendar you just created, which means you would have one such URL for each calendar you have.

Step 2: Creating the config

Paste this into your vdirsyncer config:

```
[pair holidays]
a = "holidays_public"
b = "holidays_private"
collections = null

[storage holidays_public]
type = "http"
# The URL to your iCalendar file.
url = ...

[storage holidays_private]
type = "caldav"
# The direct URL to your calendar.
url = ...
# The credentials to your CalDAV server
```

```
username = ...
password = ...
```

Then run `vdirsyncer discover holidays` and `vdirsyncer sync holidays`, and your previously created calendar should be filled with events.

Step 3: The partial_sync parameter

New in version 0.14.

You may get into a situation where you want to hide or modify some events from your `holidays` calendar. If you try to do that at this point, you'll notice that vdirsyncer will revert any changes you've made after a few times of running `sync`. This is because vdirsyncer wants to keep everything in sync, and it can't synchronize changes to the public holidays-calendar because it doesn't have the rights to do so.

For such purposes you can set the `partial_sync` parameter to `ignore`:

```
[pair holidays]
a = "holidays_public"
b = "holidays_private"
collections = null
partial_sync = ignore
```

See *the config docs* for more information.

CHAPTER 7

Full configuration manual

Vdirsyncer uses an ini-like format for storing its configuration. All values are JSON, invalid JSON will get interpreted as string:

```
x = "foo"  # String
x = foo  # Shorthand for same string

x = 42  # Integer

x = ["a", "b", "c"]  # List of strings

x = true  # Boolean
x = false

x = null  # Also known as None
```

General Section

```
[general]
status_path = ...
```

- `status_path`: A directory where vdirsyncer will store some additional data for the next sync.

  The data is needed to determine whether a new item means it has been added on one side or deleted on the other. Relative paths will be interpreted as relative to the configuration file's directory.

  See A simple synchronization algorithm for what exactly is in there.

Pair Section

```
[pair pair_name]
a = ...
b = ...
#collections = null
#conflict_resolution = null
```

- Pair names can consist of any alphanumeric characters and the underscore.
- `a` and `b` reference the storages to sync by their names.
- `collections`: A list of collections to synchronize when `vdirsyncer sync` is executed. See also *More information about collections*.

  The special values `"from a"` and `"from b"`, tell vdirsyncer to try autodiscovery on a specific storage.

  If the collection you want to sync doesn't have the same name on each side, you may also use a value of the form `["config_name", "name_a", "name_b"]`. This will synchronize the collection `name_a` on side A with the collection `name_b` on side B. The `config_name` will be used for representation in CLI arguments and logging.

  Examples:

  - `collections = ["from b", "foo", "bar"]` makes vdirsyncer synchronize the collections from side B, and also the collections named "foo" and "bar".
  - `collections = ["from b", "from a"]` makes vdirsyncer synchronize all existing collections on either side.
  - `collections = [["bar", "bar_a", "bar_b"], "foo"]` makes vdirsyncer synchronize `bar_a` from side A with `bar_b` from side B, and also synchronize `foo` on both sides with each other.
- `conflict_resolution`: Optional, define how conflicts should be handled. A conflict occurs when one item (event, task) changed on both sides since the last sync. See also *Conflict resolution*.

  Valid values are:

  - `null`, where an error is shown and no changes are done.
  - `"a wins"` and `"b wins"`, where the whole item is taken from one side.
  - `["command", "vimdiff"]`: `vimdiff <a> <b>` will be called where `<a>` and `<b>` are temporary files that contain the item of each side respectively. The files need to be exactly the same when the command returns.
    - `vimdiff` can be replaced with any other command. For example, in POSIX `["command", "cp"]` is equivalent to `"a wins"`.
    - Additional list items will be forwarded as arguments. For example, `["command", "vimdiff", "--noplugin"]` runs `vimdiff --noplugin`.

  Vdirsyncer never attempts to "automatically merge" the two items.
- `partial_sync`: Assume A is read-only, B not. If you change items on B, vdirsyncer can't sync the changes to A. What should happen instead?
  - `error`: An error is shown.
  - `ignore`: The change is ignored. However: Events deleted in B still reappear if they're updated in A.
  - `revert` (default): The change is reverted on next sync.

  See also *Syncing with read-only storages*.

- `metadata`: Metadata keys that should be synchronized when `vdirsyncer metasync` is executed. Example:

```
metadata = ["color", "displayname"]
```

This synchronizes the `color` and the `displayname` properties. The `conflict_resolution` parameter applies here as well.

Storage Section

```
[storage storage_name]
type = ...
```

- Storage names can consist of any alphanumeric characters and the underscore.
- `type` defines which kind of storage is defined. See *Supported Storages*.
- `read_only` defines whether the storage should be regarded as a read-only storage. The value `true` means synchronization will discard any changes made to the other side. The value `false` implies normal 2-way synchronization.
- Any further parameters are passed on to the storage class.

Supported Storages

CalDAV and CardDAV

caldav
CalDAV.

```
[storage example_for_caldav]
type = "caldav"
#start_date = null
#end_date = null
#item_types = []
url = "..."
#username = ""
#password = ""
#verify = true
#auth = null
#useragent = "vdirsyncer/0.16.3"
#verify_fingerprint = null
#auth_cert = null
```

You can set a timerange to synchronize with the parameters `start_date` and `end_date`. Inside those parameters, you can use any Python expression to return a valid `datetime.datetime` object. For example, the following would synchronize the timerange from one year in the past to one year in the future:

```
start_date = datetime.now() - timedelta(days=365)
end_date = datetime.now() + timedelta(days=365)
```

Either both or none have to be specified. The default is to synchronize everything.

You can set `item_types` to restrict the *kind of items* you want to synchronize. For example, if you want to only synchronize events (but don't download any tasks from the server), set `item_types = ["VEVENT"]`.

If you want to synchronize events and tasks, but have some `VJOURNAL` items on the server you don't want to synchronize, use `item_types = ["VEVENT", "VTODO"]`.

Parameters

- **start_date** – Start date of timerange to show, default -inf.
- **end_date** – End date of timerange to show, default +inf.
- **item_types** – Kind of items to show. The default, the empty list, is to show all. This depends on particular features on the server, the results are not validated.
- **url** – Base URL or an URL to a collection.
- **username** – Username for authentication.
- **password** – Password for authentication.
- **verify** – Verify SSL certificate, default True. This can also be a local path to a self-signed SSL certificate. See *SSL and certificate validation* for more information.
- **verify_fingerprint** – Optional. SHA1 or MD5 fingerprint of the expected server certificate. See *SSL and certificate validation* for more information.
- **auth** – Optional. Either `basic`, `digest` or `guess`. The default is preemptive Basic auth, sending credentials even if server didn't request them. This saves from an additional roundtrip per request. Consider setting `guess` if this causes issues with your server.
- **auth_cert** – Optional. Either a path to a certificate with a client certificate and the key or a list of paths to the files with them.
- **useragent** – Default `vdirsyncer`.

Note: Please also see *Servers*, as some servers may not work well.

---

carddav

CardDAV.

```
[storage example_for_carddav]
type = "carddav"
url = "..."
#username = ""
#password = ""
#verify = true
#auth = null
#useragent = "vdirsyncer/0.16.3"
#verify_fingerprint = null
#auth_cert = null
```

Parameters

- **url** – Base URL or an URL to a collection.
- **username** – Username for authentication.
- **password** – Password for authentication.
- **verify** – Verify SSL certificate, default True. This can also be a local path to a self-signed SSL certificate. See *SSL and certificate validation* for more information.
- **verify_fingerprint** – Optional. SHA1 or MD5 fingerprint of the expected server certificate. See *SSL and certificate validation* for more information.

- **auth** – Optional. Either `basic`, `digest` or `guess`. The default is preemptive Basic auth, sending credentials even if server didn't request them. This saves from an additional roundtrip per request. Consider setting `guess` if this causes issues with your server.
- **auth_cert** – Optional. Either a path to a certificate with a client certificate and the key or a list of paths to the files with them.
- **useragent** – Default `vdirsyncer`.

Note: Please also see *Servers*, as some servers may not work well.

Google

Vdirsyncer supports synchronization with Google calendars with the restriction that `VTODO` files are rejected by the server.

Synchronization with Google contacts is less reliable due to negligence of Google's CardDAV API. **Google's CardDAV implementation is allegedly a disaster in terms of data safety**. See this blog post for the details. Always back up your data.

At first run you will be asked to authorize application for google account access.

To use this storage type, you need to install some additional dependencies:

```
pip install vdirsyncer[google]
```

Furthermore you need to register vdirsyncer as an application yourself to obtain `client_id` and `client_secret`, as it is against Google's Terms of Service to hardcode those into opensource software *[googleterms]*:

1. Go to the Google API Manager and create a new project under any name.
2. Within that project, enable the "CalDAV" and "CardDAV" APIs (**not** the Calendar and Contacts APIs, those are different and won't work). There should be a searchbox where you can just enter those terms.
3. In the sidebar, select "Credentials" and create a new "OAuth Client ID". The application type is "Other".

   You'll be prompted to create a OAuth consent screen first. Fill out that form however you like.
4. Finally you should have a Client ID and a Client secret. Provide these in your storage config.

The `token_file` parameter should be a filepath where vdirsyncer can later store authentication-related data. You do not need to create the file itself or write anything to it.

Note: You need to configure which calendars Google should offer vdirsyncer using a rather hidden settings page.

google_calendar
Google calendar.

```
[storage example_for_google_calendar]
type = "google_calendar"
token_file = "..."
client_id = "..."
client_secret = "..."
#start_date = null
#end_date = null
#item_types = []
```

Please refer to *caldav* regarding the `item_types` and timerange parameters.

Parameters

- **token_file** – A filepath where access tokens are stored.
- **client_id/client_secret** – OAuth credentials, obtained from the Google API Manager.

google_contacts

Google contacts.

```
[storage example_for_google_contacts]
type = "google_contacts"
token_file = "..."
client_id = "..."
client_secret = "..."
```

Parameters

- **token_file** – A filepath where access tokens are stored.
- **client_id/client_secret** – OAuth credentials, obtained from the Google API Manager.

EteSync

EteSync is a new cloud provider for end to end encrypted contacts and calendar storage. Vdirsyncer contains **experimental** support for it.

To use it, you need to install some optional dependencies:

```
pip install vdirsyncer[etesync]
```

On first usage you will be prompted for the service password and the encryption password. Neither are stored.

etesync_contacts

Contacts for EteSync.

```
[storage example_for_etesync_contacts]
type = "etesync_contacts"
email = "..."
secrets_dir = "..."
#server_url = null
#db_path = null
```

Parameters

- **email** – The email address of your account.
- **secrets_dir** – A directory where vdirsyncer can store the encryption key and authentication token.
- **server_url** – Optional. URL to the root of your custom server.
- **db_path** – Optional. Use a different path for the database.

etesync_calendars

Calendars for EteSync.

```
[storage example_for_etesync_calendars]
type = "etesync_calendars"
email = "..."
secrets_dir = "..."
#server_url = null
#db_path = null
```

Parameters

- **email** – The email address of your account.
- **secrets_dir** – A directory where vdirsyncer can store the encryption key and authentication token.
- **server_url** – Optional. URL to the root of your custom server.
- **db_path** – Optional. Use a different path for the database.

Local

filesystem

Saves each item in its own file, given a directory.

```
[storage example_for_filesystem]
type = "filesystem"
path = "..."
fileext = "..."
#encoding = "utf-8"
#post_hook = null
```

Can be used with khal. See *The Vdir Storage Format* for a more formal description of the format.

Directories with a leading dot are ignored to make usage of e.g. version control easier.

Parameters

- **path** – Absolute path to a vdir/collection. If this is used in combination with the `collections` parameter in a pair-section, this should point to a directory of vdirs instead.
- **fileext** – The file extension to use (e.g. `.txt`). Contained in the href, so if you change the file extension after a sync, this will trigger a re-download of everything (but *should* not cause data-loss of any kind).
- **encoding** – File encoding for items, both content and filename.
- **post_hook** – A command to call for each item creation and modification. The command will be called with the path of the new/updated file.

singlefile

Save data in single local `.vcf` or `.ics` file.

```
[storage example_for_singlefile]
type = "singlefile"
path = "..."
#encoding = "utf-8"
```

The storage basically guesses how items should be joined in the file.

New in version 0.1.6.

Note: This storage is very slow, and that is unlikely to change. You should consider using *filesystem* if it fits your usecase.

Parameters

- **path** – The filepath to the file to be written to. If collections are used, this should contain `%s` as a placeholder for the collection name.
- **encoding** – Which encoding the file should use. Defaults to UTF-8.

Example for syncing with *caldav*:

```
[pair my_calendar]
a = my_calendar_local
b = my_calendar_remote
collections = ["from a", "from b"]

[storage my_calendar_local]
type = "singlefile"
path = ~/.calendars/%s.ics

[storage my_calendar_remote]
type = "caldav"
url = https://caldav.example.org/
#username =
#password =
```

Example for syncing with *caldav* using a `null` collection:

```
[pair my_calendar]
a = my_calendar_local
b = my_calendar_remote

[storage my_calendar_local]
type = "singlefile"
path = ~/my_calendar.ics

[storage my_calendar_remote]
type = "caldav"
url = https://caldav.example.org/username/my_calendar/
#username =
#password =
```

Read-only storages

These storages don't support writing of their items, consequently `read_only` is set to `true` by default. Changing `read_only` to `false` on them leads to an error.

http

Use a simple `.ics` file (or similar) from the web.

```
[storage example_for_http]
type = "http"
url = "..."
#username = ""
```

```
#password = ""
#verify = true
#auth = null
#useragent = "vdirsyncer/0.16.3"
#verify_fingerprint = null
#auth_cert = null
```

`webcal://`-calendars are supposed to be used with this, but you have to replace `webcal://` with `http://`, or better, `https://`.

Too many WebCAL providers generate UIDs of all `VEVENT`-components on-the-fly, i.e. all UIDs change every time the calendar is downloaded. This leads many synchronization programs to believe that all events have been deleted and new ones created, and accordingly causes a lot of unnecessary uploads and deletions on the other side. Vdirsyncer completely ignores UIDs coming from *`http`* and will replace them with a hash of the normalized item content.

Parameters

- **`url`** – URL to the `.ics` file.
- **`username`** – Username for authentication.
- **`password`** – Password for authentication.
- **`verify`** – Verify SSL certificate, default True. This can also be a local path to a self-signed SSL certificate. See *SSL and certificate validation* for more information.
- **`verify_fingerprint`** – Optional. SHA1 or MD5 fingerprint of the expected server certificate. See *SSL and certificate validation* for more information.
- **`auth`** – Optional. Either `basic`, `digest` or `guess`. The default is preemptive Basic auth, sending credentials even if server didn't request them. This saves from an additional roundtrip per request. Consider setting `guess` if this causes issues with your server.
- **`auth_cert`** – Optional. Either a path to a certificate with a client certificate and the key or a list of paths to the files with them.
- **`useragent`** – Default `vdirsyncer`.

A simple example:

```
[pair holidays]
a = holidays_local
b = holidays_remote
collections = null

[storage holidays_local]
type = "filesystem"
path = ~/.config/vdir/calendars/holidays/
fileext = .ics

[storage holidays_remote]
type = "http"
url = https://example.com/holidays_from_hicksville.ics
```

CHAPTER 8

Other tutorials

The following section contains tutorials not explicitly about any particular core function of vdirsyncer. They usually show how to integrate vdirsyncer with third-party software. Because of that, it may be that the information regarding that other software only applies to specific versions of them.

Note: Please *contribute* your own tutorials too! Pages are often only stubs and are lacking full examples.

Client applications

Vdirsyncer with Claws Mail

First of all, Claws-Mail only supports **read-only** functions for vCards. It can only read contacts, but there's no editor.

Preparation

We need to install vdirsyncer, for that look *here*. Then we need to create some folders:

```
mkdir ~/.vdirsyncer
mkdir ~/.contacts
```

Configuration

Now we create the configuration for vdirsyncer. Open `~/.vdirsyncer/config` with a text editor. The config should look like this:

```
[general]
status_path = "~/.vdirsyncer/status/"
```

```
[storage local]
type = "singlefile"
path = "~/.contacts/%s.vcf"

[storage online]
type = "carddav"
url = "CARDDAV_LINK"
username = "USERNAME"
password = "PASSWORD"
read_only = true

[pair contacts]
a = "local"
b = "online"
collections = ["from a", "from b"]
conflict_resolution = "b wins"
```

- In the general section, we define the status folder path, for discovered collections and generally stuff that needs to persist between syncs.
- In the local section we define that all contacts should be sync in a single file and the path for the contacts.
- In the online section you must change the url, username and password to your setup. We also set the storage to read-only such that no changes get synchronized back. Claws-Mail should not be able to do any changes anyway, but this is one extra safety step in case files get corrupted or vdirsyncer behaves eratically. You can leave that part out if you want to be able to edit those files locally.
- In the last section we configure that online contacts win in a conflict situation. Configure this part however you like. A correct value depends on which side is most likely to be up-to-date.

Sync

Now we discover and sync our contacts:

```
vdirsyncer discover contacts
vdirsyncer sync contacts
```

Claws Mail

Open Claws-Mail. Got to **Tools => Addressbook**.

Click on **Addressbook => New vCard**. Choose a name for the book.

Then search for the for the vCard in the folder **~/.contacts/**. Click ok, and you we will see your contacts.

Note: Claws-Mail shows only contacts that have a mail address.

Crontab

On the end we create a crontab, so that vdirsyncer syncs automatically every 30 minutes our contacts:

```
contab -e
```

On the end of that file enter this line:

```
*/30 * * * * /usr/local/bin/vdirsyncer sync > /dev/null
```

And you're done!

Running as a systemd.timer

vdirsyncer includes unit files to run at an interval (by default every 15±5 minutes).

Note: These are not installed when installing via pip, only via distribution packages. If you installed via pip, or your distribution doesn't ship systemd unit files, you'll need to download vdirsyncer.service and vdirsyncer.timer into either `/etc/systemd/user/` or `~/.local/share/systemd/user`.

Activation

To activate the timer, just run `systemctl --user enable vdirsyncer.timer`. To see logs of previous runs, use `journalctl --user -u vdirsyncer`.

Configuration

It's quite possible that the default "every fifteen minutes" interval isn't to your liking. No default will suit everybody, but this is configurable by simply running:

```
systemctl --user edit vdirsyncer
```

This will open a blank editor, where you can override the timer by including:

```
OnBootSec=5m  # This is how long after boot the first run takes place.
OnUnitActiveSec=15m  # This is how often subsequent runs take place.
```

Todoman

The iCalendar format also supports saving tasks in form of `VTODO`-entries, with the same file extension as normal events: `.ics`. Many CalDAV servers support synchronizing tasks, vdirsyncer does too.

todoman is a CLI task manager supporting *vdir*. Its interface is similar to the ones of Taskwarrior or the todo.txt CLI app. You can use *`filesystem`* with it.

Further applications, with missing pages:

- khal, a CLI calendar application supporting *vdir*. You can use *`filesystem`* with it.
- Many graphical calendar apps such as dayplanner, Orage or rainlendar save a calendar in a single `.ics` file. You can use *`singlefile`* with those.
- khard, a commandline addressbook supporting *vdir*. You can use *`filesystem`* with it.
- contactquery.c, a small program explicitly written for querying vdirs from mutt.
- mates, a commandline addressbook supporting *vdir*.
- vdirel, access *vdir* contacts from Emacs.

Servers

Baikal

Vdirsyncer is continuously tested against the latest version of Baikal.

- Baikal up to `0.2.7` also uses an old version of SabreDAV, with the same issue as ownCloud, see issue #160. This issue is fixed in later versions.

DavMail (Exchange, Outlook)

DavMail is a proxy program that allows you to use Card- and CalDAV clients with Outlook. That allows you to use vdirsyncer with Outlook.

In practice your success with DavMail may wildly vary. Depending on your Exchange server you might get confronted with weird errors of all sorts (including data-loss).

Make absolutely sure you use the latest DavMail:

```
[storage outlook]
type = "caldav"
url = "http://localhost:1080/users/user@example.com/calendar/"
username = "user@example.com"
password = ...
```

- Older versions of DavMail handle URLs case-insensitively. See issue #144.
- DavMail is handling malformed data on the Exchange server very poorly. In such cases the Calendar Checking Tool for Outlook might help.
- In some cases, you may see errors about duplicate events. It may look something like this:

```
error: my_calendar/calendar: Storage "my_calendar_remote/calendar" contains␣
↪multiple items with the same UID or even content. Vdirsyncer will now abort the␣
↪synchronization of this collection, because the fix for this is not clear; It␣
↪could be the result of a badly behaving server. You can try running:
error:
error:     vdirsyncer repair my_calendar_remote/calendar
error:
error: But make sure to have a backup of your data in some form. The offending␣
↪hrefs are:
[...]
```

  In order to fix this, you can try the Remove-DuplicateAppointments.ps1 PowerShell script that Microsoft has come up with in order to remove duplicates.

FastMail

Vdirsyncer is continuously tested against FastMail, thanks to them for providing a free account for this purpose. There are no known issues with it. FastMail's support pages provide the settings to use:

```
[storage cal]
type = "caldav"
url = "https://caldav.messagingengine.com/"
username = ...
password = ...
```

```
[storage card]
type = "carddav"
url = "https://carddav.messagingengine.com/"
username = ...
password = ...
```

Google

Using vdirsyncer with Google Calendar is possible as of 0.10, but it is not tested frequently. You can use *google_contacts* and *google_calendar*.

For more information see issue #202 and issue #8.

iCloud

Vdirsyncer is regularly tested against iCloud.

```
[storage cal]
type = "caldav"
url = "https://caldav.icloud.com/"
username = ...
password = ...

[storage card]
type = "carddav"
url = "https://contacts.icloud.com/"
username = ...
password = ...
```

Problems:

- Vdirsyncer can't do two-factor auth with iCloud (there doesn't seem to be a way to do two-factor auth over the DAV APIs) You'll need to use app-specific passwords instead.
- iCloud has a few special requirements when creating collections. In principle vdirsyncer can do it, but it is recommended to create them from an Apple client (or the iCloud web interface).
  - iCloud requires a minimum length of collection names.
  - Calendars created by vdirsyncer cannot be used as tasklists.

nextCloud

Vdirsyncer is continuously tested against the latest version of nextCloud:

```
[storage cal]
type = "caldav"
url = "https://nextcloud.example.com/"
username = ...
password = ...

[storage card]
type = "carddav"
url = "https://nextcloud.example.com/"
```

- WebCAL-subscriptions can't be discovered by vdirsyncer. See this relevant issue.

ownCloud

Vdirsyncer is continuously tested against the latest version of ownCloud:

```
[storage cal]
type = "caldav"
url = "https://example.com/remote.php/dav/"
username = ...
password = ...

[storage card]
type = "carddav"
url = "https://example.com/remote.php/dav/"
username = ...
password = ...
```

- *Versions older than 7.0.0:* ownCloud uses SabreDAV, which had problems detecting collisions and race-conditions. The problems were reported and are fixed in SabreDAV's repo, and the corresponding fix is also in ownCloud since 7.0.0. See issue #16 for more information.

Radicale

Radicale is a very lightweight server, however, it intentionally doesn't implement the CalDAV and CardDAV standards completely, which might lead to issues even with very well-written clients. Apart from its non-conformity with standards, there are multiple other problems with its code quality and the way it is maintained. Consider using e.g. *Xandikos* instead.

That said, vdirsyncer is continuously tested against the git version and the latest PyPI release of Radicale.

- Vdirsyncer can't create collections on Radicale.
- Radicale doesn't support time ranges in the calendar-query of CalDAV, so setting `start_date` and `end_date` for *`caldav`* will have no or unpredicted consequences.
- Versions of Radicale older than 0.9b1 choke on RFC-conform queries for all items of a collection.

  You have to set `item_types = ["VTODO", "VEVENT"]` in *`caldav`* for vdirsyncer to work with those versions.

Xandikos

Xandikos is a lightweight, yet complete CalDAV and CardDAV server, backed by git. Vdirsyncer is continuously tested against its latest version.

After running `./bin/xandikos --defaults -d $HOME/dav`, you should be able to point vdirsyncer against the root of Xandikos like this:

```
[storage cal]
type = "caldav"
url = "https://xandikos.example.com/"
username = ...
password = ...

[storage card]
```

```
type = "carddav"
url = "https://xandikos.example.com/"
username = ...
password = ...
```

CHAPTER 9

Known Problems

For any unanswered questions or problems, see *Support and Contact*.

Requests-related ImportErrors

> ImportError: No module named packages.urllib3.poolmanager
>
> ImportError: cannot import name iter_field_objects

Debian and nowadays even other distros make modifications to the `requests` package that don't play well with packages assuming a normal `requests`. This is due to stubbornness on both sides.

See issue #82 and issue #140 for past discussions. You have one option to work around this, that is, to install vdirsyncer in a virtualenv, see *Manual installation*.

CHAPTER 10

Contributing to this project

Note:

- Please read *Support and Contact* for questions and support requests.
- All participants must follow the pimutils Code of Conduct.

The issue tracker

We use GitHub issues for organizing bug reports and feature requests.

The following labels are of interest:

- "Planning" is for issues that are still undecided, but where at least some discussion exists.
- "Blocked" is for issues that can't be worked on at the moment because some other unsolved problem exists. This problem may be a bug in some software dependency, for instance.
- "Ready" contains issues that are ready to work on.

If you just want to get started with contributing, the "ready" issues are an option. Issues that are still in "Planning" are also an option, but require more upfront thinking and may turn out to be impossible to solve, or at least harder than anticipated. On the flip side those tend to be the more interesting issues as well, depending on how one looks at it.

All of those labels are also available as a kanban board on waffle.io. It is really just an alternative overview over all issues, but might be easier to comprehend.

Feel free to *contact* me or comment on the relevant issues for further information.

Reporting bugs

- Make sure your problem isn't already listed in *Known Problems*.

- Make sure you have the absolutely latest version of vdirsyncer. For users of some Linux distributions such as Debian or Fedora this may not be the version that your distro offers. In those cases please file a bug against the distro package, not against upstream vdirsyncer.
- Use `--verbosity=DEBUG` when including output from vdirsyncer.

Suggesting features

If you're suggesting a feature, keep in mind that vdirsyncer tries not to be a full calendar or contacts client, but rather just the piece of software that synchronizes all the data. *Take a look at the documentation for software working with vdirsyncer*.

Submitting patches, pull requests

- **Discuss everything in the issue tracker first** (or contact me somehow else) before implementing it.
- Make sure the tests pass. See below for running them.
- But not because you wrote too few tests.
- Add yourself to `AUTHORS.rst`, and add a note to `CHANGELOG.rst` too.

Running tests, how to set up your development environment

For many patches, it might suffice to just let Travis run the tests. However, Travis is slow, so you might want to run them locally too. For this, set up a virtualenv and run this inside of it:

```
# install:
#  - vdirsyncer from the repo into the virtualenv
#  - stylecheckers (flake8) and code formatters (autopep8)
make install-dev

# Install git commit hook for the stylechecker
make install-git-hooks

# install test dependencies
make install-test
```

Then you can run:

```
make test   # The normal testsuite
make style  # Stylechecker
make docs   # Build the HTML docs, output is at docs/_build/html/
```

The `Makefile` has a lot of options that allow you to control which tests are run, and which servers are tested. Take a look at its code where they are all initialized and documented.

For example, to test xandikos, run:

```
make DAV_SERVER=xandikos install-test
make DAV_SERVER=xandikos test
```

If you have any questions, feel free to open issues about it.

Structure of the testsuite

Within `tests/`, there are three main folders:

- `system` contains system- and also integration tests. A rough rule is: If the test is using temporary files, put it here.
- `unit`, where each testcase tests a single class or function.
- `storage` runs a generic storage testsuite against all storages.

The reason for this separation is: We are planning to generate separate coverage reports for each of those testsuites. Ideally `unit` would generate palatable coverage of the entire codebase *on its own*, and the *combination* of `system` and `storage` as well.

CHAPTER 11

The Vdir Storage Format

This document describes a standard for storing calendars and contacts on a filesystem, with the main goal of being easy to implement.

Vdirsyncer synchronizes to vdirs via *filesystem*. Each vdir (basically just a directory with some files in it) represents a calendar or addressbook.

Basic Structure

The main folder (root) contains an arbitrary number of subfolders (collections), which contain only files (items). Synonyms for "collection" may be "addressbook" or "calendar".

An item is:

- A vCard file, in which case the file extension *must* be *.vcf*, *or*
- An iCalendar file, in which case the file extension *must* be *.ics*.

An item *should* contain a `UID` property as described by the vCard and iCalendar standards. If it contains more than one `UID` property, the values of those *must* not differ.

The file *must* contain exactly one event, task or contact. In most cases this also implies only one `VEVENT`/`VTODO`/`VCARD` component per file, but e.g. recurrence exceptions would require multiple `VEVENT` components per event.

The filename should have similar properties as the `UID` of the file content. However, there is no requirement for these two to be the same. Programs may choose to store additional metadata in that filename, however, at the same time they *must not* assume that the metadata they included will be preserved by other programs.

Metadata

Any of the below metadata files may be absent. None of the files listed below have any file extensions.

- A file called `color` inside the vdir indicates the vdir's color, a property that is only relevant in UI design.

  Its content is an ASCII-encoded hex-RGB value of the form `#RRGGBB`. For example, a file content of `#FF0000` indicates that the vdir has a red (user-visible) color. No short forms or informal values such as `red` (as known from CSS, for example) are allowed. The prefixing `#` must be present.

- A file called `displayname` contains a UTF-8 encoded label that may be used to represent the vdir in UIs.

Writing to vdirs

Creating and modifying items or metadata files *should* happen atomically.

Writing to a temporary file on the same physical device, and then moving it to the appropriate location is usually a very effective solution. For this purpose, files with the extension `.tmp` may be created inside collections.

When changing an item, the original filename *must* be used.

Reading from vdirs

- Any file ending with the `.tmp` or no file extension *must not* be treated as an item.
- The `ident` part of the filename *should not* be parsed to improve the speed of item lookup.

Considerations

The primary reason this format was chosen is due to its compatibility with the CardDAV and CalDAV standards.

Performance

Currently, vdirs suffer from a rather major performance problem, one which current implementations try to mitigate by building up indices of the collections for faster search and lookup.

The reason items' filenames don't contain any extra information is simple: The solutions presented induced duplication of data, where one duplicate might become out of date because of bad implementations. As it stands right now, a index format could be formalized separately though.

vdirsyncer doesn't really have to bother about efficient item lookup, because its synchronization algorithm needs to fetch the whole list of items anyway. Detecting changes is easily implemented by checking the files' modification time.

CHAPTER 12

Packaging guidelines

Thank you very much for packaging vdirsyncer! The following guidelines should help you to avoid some common pitfalls.

While they are called guidelines and therefore theoretically not mandatory, if you consider going a different direction, please first open an issue or contact me otherwise instead of just going ahead. These guidelines exist for my own convenience too.

Obtaining the source code

The main distribution channel is PyPI, and source tarballs can be obtained there. Do not use the ones from GitHub: Their tarballs contain useless junk and are more of a distraction than anything else.

I give each release a tag in the git repo. If you want to get notified of new releases, GitHub's feed is a good way.

Dependency versions

As with most Python packages, `setup.py` denotes the dependencies of vdirsyncer. It also contains lower-bound versions of each dependency. Older versions will be rejected by the testsuite.

Testing

Everything testing-related goes through the `Makefile` in the root of the repository or PyPI package. Trying to e.g. run `py.test` directly will require a lot of environment variables to be set (for configuration) and you probably don't want to deal with that.

You can install the testing dependencies with:

```
make install-test
```

You probably don't want this since it will use pip to download the dependencies. Alternatively you can find the testing dependencies in `test-requirements.txt`, again with lower-bound version requirements.

You also have to have vdirsyncer fully installed at this point. Merely `cd`-ing into the tarball will not be sufficient.

Running the tests happens with:

```
make test
```

Hypothesis will randomly generate test input. If you care about deterministic tests, set the `DETERMINISTIC_TESTS` variable to `"true"`:

```
make DETERMINISTIC_TESTS=true test
```

There are a lot of additional variables that allow you to test vdirsyncer against a particular server. Those variables are not "stable" and may change drastically between minor versions. Just don't use them, you are unlikely to find bugs that vdirsyncer's CI hasn't found.

Documentation

Using Sphinx you can generate the documentation you're reading right now in a variety of formats, such as HTML, PDF, or even as a manpage. That said, I only take care of the HTML docs' formatting.

You can find a list of dependencies in `docs-requirements.txt`. Again, you can install those using pip with:

```
make install-docs
```

Then change into the `docs/` directory and build whatever format you want using the `Makefile` in there (run `make` for the formats you can build).

Contrib files

Reference `systemd.service` and `systemd.timer` unit files are provided. It is recommended to install this if your distribution is systemd-based.

CHAPTER 13

Support and Contact

- The `#pimutils` IRC channel on Freenode might be active, depending on your timezone. Use it for support and general (including off-topic) discussion.
- Open a GitHub issue for concrete bug reports and feature requests.
- Lastly, you can also contact the author directly. Do this for security issues. If that doesn't work out (i.e. if I don't respond within one week), use `contact@pimutils.org`.

CHAPTER 14

Changelog

This changelog only contains information that might be useful to end users and package maintainers. For further info, see the git commit log.

Package maintainers and users who have to manually update their installation may want to subscribe to GitHub's tag feed.

Version 0.16.3

released on 03 October 2017

- First version with custom Debian and Ubuntu packages. See issue #663.
- Remove invalid ASCII control characters from server responses. See issue #626.
- **packagers:** Python 3.3 is no longer supported. See pull request #674.

Version 0.16.2

released on 24 August 2017

- Fix crash when using daterange or item_type filters in `google_calendar`, see issue #657.
- **Packagers:** Fixes for new version `0.2.0` of `click-log`. The version requirements for the dependency `click-log` changed.

Version 0.16.1

released on 8 August 2017

- Removed remoteStorage support, see issue #647.

- Fixed test failures caused by latest requests version, see issue #660.

Version 0.16.0

released on 2 June 2017

- Strip `METHOD:PUBLISH` added by some calendar providers, see issue #502.
- Fix crash of Google storages when saving token file.
- Make DAV discovery more RFC-conformant, see pull request #585.
- Vdirsyncer is now tested against Xandikos, see pull request #601.
- Subfolders with a leading dot are now ignored during discover for `filesystem` storage. This makes it easier to combine it with version control.
- Statuses are now stored in a sqlite database. Old data is automatically migrated. Users with really large datasets should encounter performance improvements. This means that **sqlite3 is now a dependency of vdirsyncer**.
- **Vdirsyncer is now licensed under the 3-clause BSD license**, see issue #610.
- Vdirsyncer now includes experimental support for EteSync, see pull request #614.
- Vdirsyncer now uses more filesystem metadata for determining whether an item changed. You will notice a **possibly heavy CPU/IO spike on the first sync after upgrading**.
- **Packagers:** Reference `systemd.service` and `systemd.timer` unit files are provided. It is recommended to install these as documentation if your distribution is systemd-based.

Version 0.15.0

released on 28 February 2017

- Deprecated syntax for configuration values is now completely rejected. All values now have to be valid JSON.
- A few UX improvements for Google storages, see issue #549 and issue #552.
- Fix collection discovery for *`google_contacts`*, see issue #564.
- iCloud is now tested on Travis, see issue #567.

Version 0.14.1

released on 05 January 2017

- `vdirsyncer repair` no longer changes "unsafe" UIDs by default, an extra option has to be specified. See issue #527.
- A lot of important documentation updates.

Version 0.14.0

released on 26 October 2016

- `vdirsyncer sync` now continues other uploads if one upload failed. The exit code in such situations is still non-zero.
- Add `partial_sync` option to pair section. See *the config docs*.
- Vdirsyner will now warn if there's a string without quotes in your config. Please file issues if you find documentation that uses unquoted strings.
- Fix an issue that would break khal's config setup wizard.

Version 0.13.1

released on 30 September 2016

- Fix a bug that would completely break collection discovery.

Version 0.13.0

released on 29 September 2016

- Python 2 is no longer supported at all. See issue #219.
- Config sections are now checked for duplicate names. This also means that you cannot have a storage section `[storage foo]` and a pair `[pair foo]` in your config, they have to have different names. This is done such that console output is always unambiguous. See issue #459.
- Custom commands can now be used for conflict resolution during sync. See issue #127.
- *`http`* now completely ignores UIDs. This avoids a lot of unnecessary down- and uploads.

Version 0.12.1

released on 20 August 2016

- Fix a crash for Google and DAV storages. See pull request #492.
- Fix an URL-encoding problem with DavMail. See issue #491.

Version 0.12

released on 19 August 2016

- *`singlefile`* now supports collections. See pull request #488.

Version 0.11.3

released on 29 July 2016

- Default value of `auth` parameter was changed from `guess` to `basic` to resolve issues with the Apple Calendar Server (issue #457) and improve performance. See issue #461.
- **Packagers:** The `click-threading` requirement is now `>=0.2`. It was incorrect before. See issue #478.

- Fix a bug in the DAV XML parsing code that would make vdirsyncer crash on certain input. See issue #480.
- Redirect chains should now be properly handled when resolving `well-known` URLs. See pull request #481.

Version 0.11.2

released on 15 June 2016

- Fix typo that would break tests.

Version 0.11.1

released on 15 June 2016

- Fix a bug in collection validation.
- Fix a cosmetic bug in debug output.
- Various documentation improvements.

Version 0.11.0

released on 19 May 2016

- Discovery is no longer automatically done when running `vdirsyncer sync`. `vdirsyncer discover` now has to be explicitly called.
- Add a `.plist` example for Mac OS X.
- Usage under Python 2 now requires a special config parameter to be set.
- Various deprecated configuration parameters do no longer have specialized errormessages. The generic error message for unknown parameters is shown.
  - Vdirsyncer no longer warns that the `passwordeval` parameter has been renamed to `password_command`.
  - The `keyring` fetching strategy has been dropped some versions ago, but the specialized error message has been dropped.
  - An old status format from version 0.4 is no longer supported. If you're experiencing problems, just delete your status folder.

Version 0.10.0

released on 23 April 2016

- New storage types *`google_calendar`* and *`google_contacts`* have been added.
- New global command line option *–config*, to specify an alternative config file. See issue #409.
- The `collections` parameter can now be used to synchronize differently-named collections with each other.
- **Packagers:** The `lxml` dependency has been dropped.
- XML parsing is now a lot stricter. Malfunctioning servers that used to work with vdirsyncer may stop working.

Version 0.9.3

released on 22 March 2016

- *singlefile* and *http* now handle recurring events properly.
- Fix a typo in the packaging guidelines.
- Moved to `pimutils` organization on GitHub. Old links *should* redirect, but be aware of client software that doesn't properly handle redirects.

Version 0.9.2

released on 13 March 2016

- Fixed testsuite for environments that don't have any web browser installed. See pull request #384.

Version 0.9.1

released on 13 March 2016

- Removed leftover debug print statement in `vdirsyncer discover`, see commit `3d856749f37639821b148238ef35f1acba82db36`.
- `metasync` will now strip whitespace from the start and the end of the values. See issue #358.
- New `Packaging Guidelines` have been added to the documentation.

Version 0.9.0

released on 15 February 2016

- The `collections` parameter is now required in pair configurations. Vdirsyncer will tell you what to do in its error message. See issue #328.

Version 0.8.1

released on 30 January 2016

- Fix error messages when invalid parameter fetching strategy is used. This is important because users would receive awkward errors for using deprecated `keyring` fetching.

Version 0.8.0

released on 27 January 2016

- Keyring support has been removed, which means that `password.fetch = ["keyring", "example.com", "myuser"]` doesn't work anymore.

  For existing setups: Use `password.fetch = ["command", "keyring", "get", "example.com", "myuser"]` instead, which is more generic. See the documentation for details.

- Now emitting a warning when running under Python 2. See issue #219.

Version 0.7.5

released on 23 December 2015

- Fixed a bug in `remotestorage` that would try to open a CLI browser for OAuth.
- Fix a packaging bug that would prevent vdirsyncer from working with newer lxml versions.

Version 0.7.4

released on 22 December 2015

- Improved error messages instead of faulty server behavior, see issue #290 and issue #300.
- Safer shutdown of threadpool, avoid exceptions, see issue #291.
- Fix a sync bug for read-only storages see commit `ed22764921b2e5bf6a934cf14aa9c5fede804d8e`.
- Etag changes are no longer sufficient to trigger sync operations. An actual content change is also necessary. See issue #257.
- `remotestorage` now automatically opens authentication dialogs in your configured GUI browser.
- **Packagers:** `lxml>=3.1` is now required (newer lower-bound version).

Version 0.7.3

released on 05 November 2015

- Make remotestorage-dependencies actually optional.

Version 0.7.2

released on 05 November 2015

- Un-break testsuite.

Version 0.7.1

released on 05 November 2015

- **Packagers:** The setuptools extras `keyring` and `remotestorage` have been added. They're basically optional dependencies. See `setup.py` for more details.
- Highly experimental remoteStorage support has been added. It may be completely overhauled or even removed in any version.
- Removed mentions of old `password_command` in documentation.

Version 0.7.0

released on 27 October 2015

- **Packagers:** New dependencies are `click_threading`, `click_log` and `click>=5.0`.
- `password_command` is gone. Keyring support got completely overhauled. See *Storing passwords*.

Version 0.6.0

released on 06 August 2015

- `password_command` invocations with non-zero exit code are now fatal (and will abort synchronization) instead of just producing a warning.
- Vdirsyncer is now able to synchronize metadata of collections. Set `metadata = ["displayname"]` and run `vdirsyncer metasync`.
- **Packagers:** Don't use the GitHub tarballs, but the PyPI ones.
- **Packagers:** `build.sh` is gone, and `Makefile` is included in tarballs. See the content of `Makefile` on how to run tests post-packaging.
- `verify_fingerprint` doesn't automatically disable `verify` anymore.

Version 0.5.2

released on 15 June 2015

- Vdirsyncer now checks and corrects the permissions of status files.
- Vdirsyncer is now more robust towards changing UIDs inside items.
- Vdirsyncer is now handling unicode hrefs and UIDs correctly. Software that produces non-ASCII UIDs is broken, but apparently it exists.

Version 0.5.1

released on 29 May 2015

- **N.b.: The PyPI upload of 0.5.0 is completely broken.**
- Raise version of required requests-toolbelt to `0.4.0`.
- Command line should be a lot faster when no work is done, e.g. for help output.
- Fix compatibility with iCloud again.
- Use only one worker if debug mode is activated.
- `verify=false` is now disallowed in vdirsyncer, please use `verify_fingerprint` instead.
- Fixed a bug where vdirsyncer's DAV storage was not using the configured useragent for collection discovery.

Version 0.4.4

released on 12 March 2015

- Support for client certificates via the new `auth_cert` parameter, see issue #182 and pull request #183.
- The `icalendar` package is no longer required.
- Several bugfixes related to collection creation.

Version 0.4.3

released on 20 February 2015

- More performance improvements to `singlefile`-storage.
- Add `post_hook` param to `filesystem`-storage.
- Collection creation now also works with SabreDAV-based servers, such as Baikal or ownCloud.
- Removed some workarounds for Radicale. Upgrading to the latest Radicale will fix the issues.
- Fixed issues with iCloud discovery.
- Vdirsyncer now includes a simple `repair` command that seeks to fix some broken items.

Version 0.4.2

released on 30 January 2015

- Vdirsyncer now respects redirects when uploading and updating items. This might fix issues with Zimbra.
- Relative `status_path` values are now interpreted as relative to the configuration file's directory.
- Fixed compatibility with custom SabreDAV servers. See issue #166.
- Catch harmless threading exceptions that occur when shutting down vdirsyncer. See issue #167.
- Vdirsyncer now depends on `atomicwrites`.
- Massive performance improvements to `singlefile`-storage.
- Items with extremely long UIDs should now be saved properly in `filesystem`-storage. See issue #173.

Version 0.4.1

released on 05 January 2015

- All `create` arguments from all storages are gone. Vdirsyncer now asks if it should try to create collections.
- The old config values `True`, `False`, `on`, `off` and `None` are now invalid.
- UID conflicts are now properly handled instead of ignoring one item. Card- and CalDAV servers are already supposed to take care of those though.
- Official Baikal support added.

Version 0.4.0

released on 31 December 2014

- The `passwordeval` parameter has been renamed to `password_command`.
- The old way of writing certain config values such as lists is now gone.
- Collection discovery has been rewritten. Old configuration files should be compatible with it, but vdirsyncer now caches the results of the collection discovery. You have to run `vdirsyncer discover` if collections were added or removed on one side.
- Pair and storage names are now restricted to certain characters. Vdirsyncer will issue a clear error message if your configuration file is invalid in that regard.
- Vdirsyncer now supports the XDG-Basedir specification. If the `VDIRSYNCER_CONFIG` environment variable isn't set and the `~/.vdirsyncer/config` file doesn't exist, it will look for the configuration file at `$XDG_CONFIG_HOME/vdirsyncer/config`.
- Some improvements to CardDAV and CalDAV discovery, based on problems found with FastMail. Support for `.well-known`-URIs has been added.

Version 0.3.4

released on 8 December 2014

- Some more bugfixes to config handling.

Version 0.3.3

released on 8 December 2014

- Vdirsyncer now also works with iCloud. Particularly collection discovery and etag handling were fixed.
- Vdirsyncer now encodes Cal- and CardDAV requests differently. This hasn't been well-tested with servers like Zimbra or SoGo, but isn't expected to cause any problems.
- Vdirsyncer is now more robust regarding invalid responses from CalDAV servers. This should help with future compatibility with Davmail/Outlook.
- Fix a bug when specifying `item_types` of *`caldav`* in the deprecated config format.
- Fix a bug where vdirsyncer would ignore all but one character specified in `unsafe_href_chars` of *`caldav`* and *`carddav`*.

Version 0.3.2

released on 3 December 2014

- The current config format has been deprecated, and support for it will be removed in version 0.4.0. Vdirsyncer warns about this now.

Version 0.3.1

released on 24 November 2014

- Fixed a bug where vdirsyncer would delete items if they're deleted on side A but modified on side B. Instead vdirsyncer will now upload the new items to side A. See issue #128.
- Synchronization continues with the remaining pairs if one pair crashes, see issue #121.
- The `processes` config key is gone. There is now a `--max-workers` option on the CLI which has a similar purpose. See pull request #126.
- The Read The Docs-theme is no longer required for building the docs. If it is not installed, the default theme will be used. See issue #134.

Version 0.3.0

released on 20 September 2014

- Add `verify_fingerprint` parameter to *`http`*, *`caldav`* and *`carddav`*, see issue #99 and pull request #106.
- Add `passwordeval` parameter to *General Section*, see issue #108 and pull request #117.
- Emit warnings (instead of exceptions) about certain invalid responses from the server, see issue #113. This is apparently required for compatibility with Davmail.

Version 0.2.5

released on 27 August 2014

- Don't ask for the password of one server more than once and fix multiple concurrency issues, see issue #101.
- Better validation of DAV endpoints.

Version 0.2.4

released on 18 August 2014

- Include workaround for collection discovery with latest version of Radicale.
- Include metadata files such as the changelog or license in source distribution, see issue #97 and issue #98.

Version 0.2.3

released on 11 August 2014

- Vdirsyncer now has a `--version` flag, see issue #92.
- Fix a lot of bugs related to special characters in URLs, see issue #49.

Version 0.2.2

released on 04 August 2014

- Remove a security check that caused problems with special characters in DAV URLs and certain servers. On top of that, the security check was nonsensical. See issue #87 and issue #91.
- Change some errors to warnings, see issue #88.
- Improve collection autodiscovery for servers without full support.

Version 0.2.1

released on 05 July 2014

- Fix bug where vdirsyncer shows empty addressbooks when using CardDAV with Zimbra.
- Fix infinite loop when password doesn't exist in system keyring.
- Colorized errors, warnings and debug messages.
- vdirsyncer now depends on the `click` package instead of argvard.

Version 0.2.0

released on 12 June 2014

- vdirsyncer now depends on the `icalendar` package from PyPI, to get rid of its own broken parser.
- vdirsyncer now also depends on `requests_toolbelt`. This makes it possible to guess the authentication type instead of blankly assuming `basic`.
- Fix a semi-bug in caldav and carddav storages where a tuple (href, etag) instead of the proper etag would have been returned from the upload method. vdirsyncer might do unnecessary copying when upgrading to this version.
- Add the storage *`singlefile`*. See issue #48.
- The `collections` parameter for pair sections now accepts the special values `from a` and `from b` for automatically discovering collections. See *Pair Section*.
- The `read_only` parameter was added to storage sections. See *Storage Section*.

Version 0.1.5

released on 14 May 2014

- Introduced changelogs
- Many bugfixes
- Many doc fixes
- vdirsyncer now doesn't necessarily need UIDs anymore for synchronization.
- vdirsyncer now aborts if one collection got completely emptied between synchronizations. See issue #42.

CHAPTER 15

Credits and License

Contributors

In alphabetical order:

- Ben Boeckel
- Christian Geier
- Clément Mondon
- Hugo Osvaldo Barrera
- Julian Mehne
- Malte Kiefer
- Marek Marczykowski-Górecki
- Markus Unterwaditzer
- Michael Adler
- Thomas Weißschuh

Additionally FastMail sponsored a paid account for testing. Thanks!

License

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

CHAPTER 16

Donations

If you found my work useful, please consider donating. Thank you!

- Bitcoin: `16sSHxZm263WHR9P9PJjCxp64jp9ooXKVt`
- PayPal.me
- Bountysource is useful for funding work on a specific GitHub issue.
  - There's also Bountysource Salt, for one-time and recurring donations.
  - Donations via Bountysource are publicly listed. Use PayPal if you dislike that.
- Flattr or Gratipay can be used for recurring donations.

Bibliography

[googleterms] See ToS, section “Confidential Matters”.

Index