# Syncany User Guide

***Release 0.4.6-alpha***

**Philipp Heckel**

**Apr 22, 2017**

# Contents

This is the user guide for Syncany, an open-source cloud storage and filesharing application with a strong focus on security.

This document is targeted at end-users. It describes how to use Syncany and how to configure it. For development and build instructions, please refer to the GitHub page or the project wiki.

**Contents**

CHAPTER 1

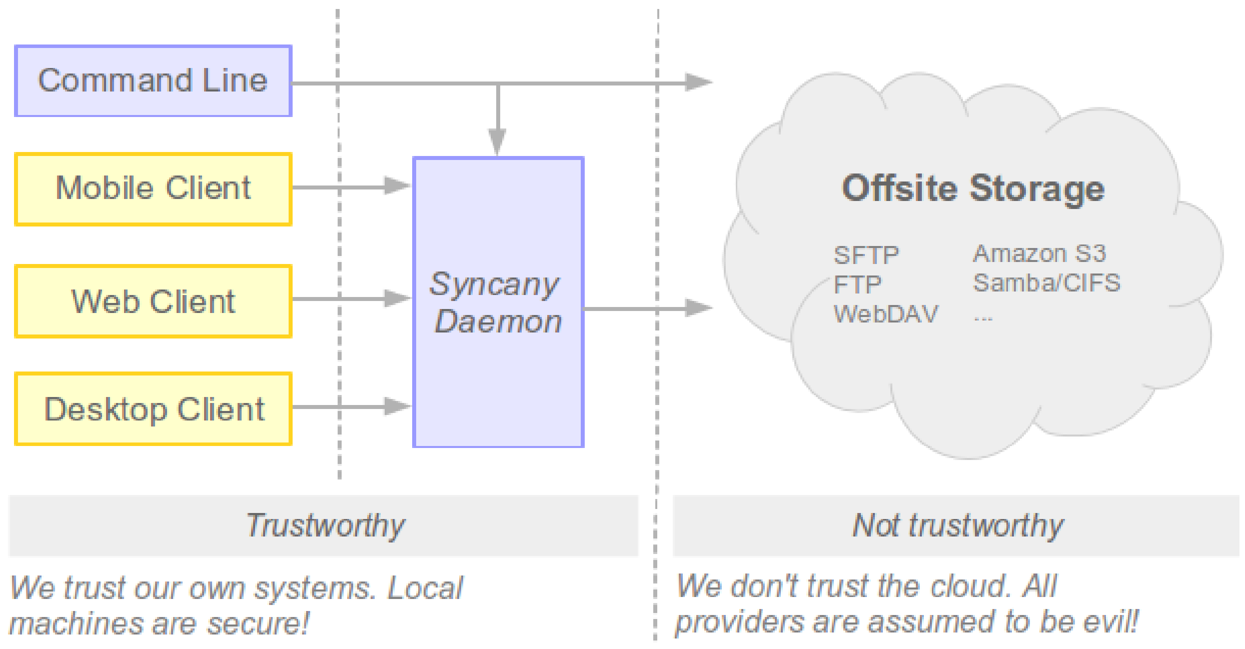
# What is Syncany?

Syncany is an open source Dropbox-like file sync and backup tool. It synchronizes files and folders between computers either manually or automatically. Users can define certain folders on their machine and keep them in sync with friends or colleagues.

So what makes Syncany **awesome**?

- Automatic/continuous file synchronization
- Use-your-own storage (FTP, S3, WebDAV, NFS, Samba/Windows file share, ...)
- Files are encrypted before upload (don't trust anybody but yourself)
- Files are deduplicated locally (massive space savings on remote storage)
- Files are intelligently versioned (restoring old versions or deleted files is easy)

**Excited yet? Ready to learn more?**



Syncany takes the idea of a Dropbox-like continuous file synchronization tool and applies it to the real world: It makes use of the storage that is *available* rather than being bound to a certain protocol or backend a certain provider is offering. **With a simple plugin-based storage system, Syncany can turn almost anything into a usable backend storage**. Right now, it can use folders on an FTP or SFTP storage, WebDAV or Samba shares and Amazon S3 buckets, but that's just the beginning. Because it's so easy to implement plugins, more are definitely coming. Even things like using IMAP folders or Flickr's free 1 TB of image data could be used to store your data.

Besides its incredible flexibilty in terms of storage, Syncany is also a very privacy-aware software: **User data never leaves the local machine without being encrypted**. Data confidentiality and user privacy are the taken very seriously – 128-bit AES+Twofish/GCM encryption is built-in by default.

Combining these two features makes Syncany really awesome: You can use all of the free storage that the Internet gives you (Google Drive, OneDrive, etc.) to share your files, but you don't have to worry about your privacy. Syncany takes care of encryption before files are uploaded.

## How do I use it?

As of today, Syncany *is available* as **a command line tool** on all major operating systems. We're working on a **graphical user interface** and a **web frontend**, but until that is ready, you'll have to make due with the command line.

You can either *manually trigger* the file synchronization like you do with Git, rsync or Unison; or you can let the Syncany background process *automatically sync your files*. Check out the *Getting Started Guide* to learn more.

## Who needs another sync tool?

There are many sync tools like Syncany out there, so depending on your use case, Syncany might just not be for you. The target audience for Syncany is not the average *cloud user* (you know, the one who puts his life in a provider's hand), but rather the more privacy-aware user and/or users/companies that want to use their own storage:

- If you want to share files with friends or colleagues without having to trust anyone but yourself, Syncany is the right tool for you.

- If you want to use your own offsite storage instead of a provider-controlled storage, Syncany is the right tool for you.

We've identified three major use cases that Syncany can be used for:

- Continuous file synchronization
- Interval-based or on-demand backups
- Simple binary-compatible file versioning

## What Syncany is not!

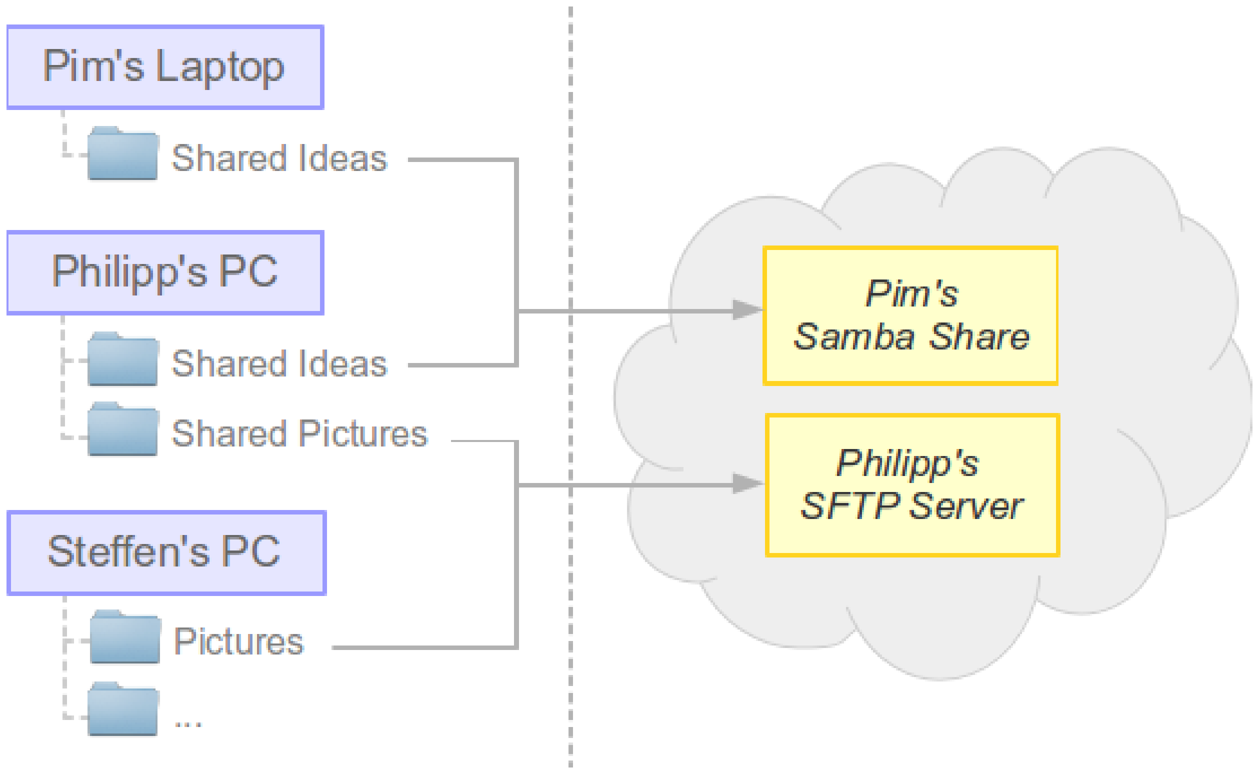
Because there are so many sync tools out there, it's very easy to assume that Syncany is just like the others. Here are a few things that Syncany is not:

- Syncany is not a free or pay-as-you-go service. Syncany is a tool.
- Syncany does not provide storage. **You must bring your own storage.**
- Syncany is not a peer-to-peer network. Data is stored centrally and you must trust the persons you share files with.
- Syncany has version control capabilities, but it is not a fully fledged version control system like Git.
- Syncany is **not yet usable for critical files**. It's alpha software. Don't do it!

## Example Use Cases

### Use Case 1: Friends sharing files

Pim wants to share some ideas with Philipp, and Philipp wants to share some pictures with Steffen. Pim has a Windows server running a Samba server (Windows share), and Philipp has has a virtual server from a random hosting company lying around – all set up with a spare SFTP account.



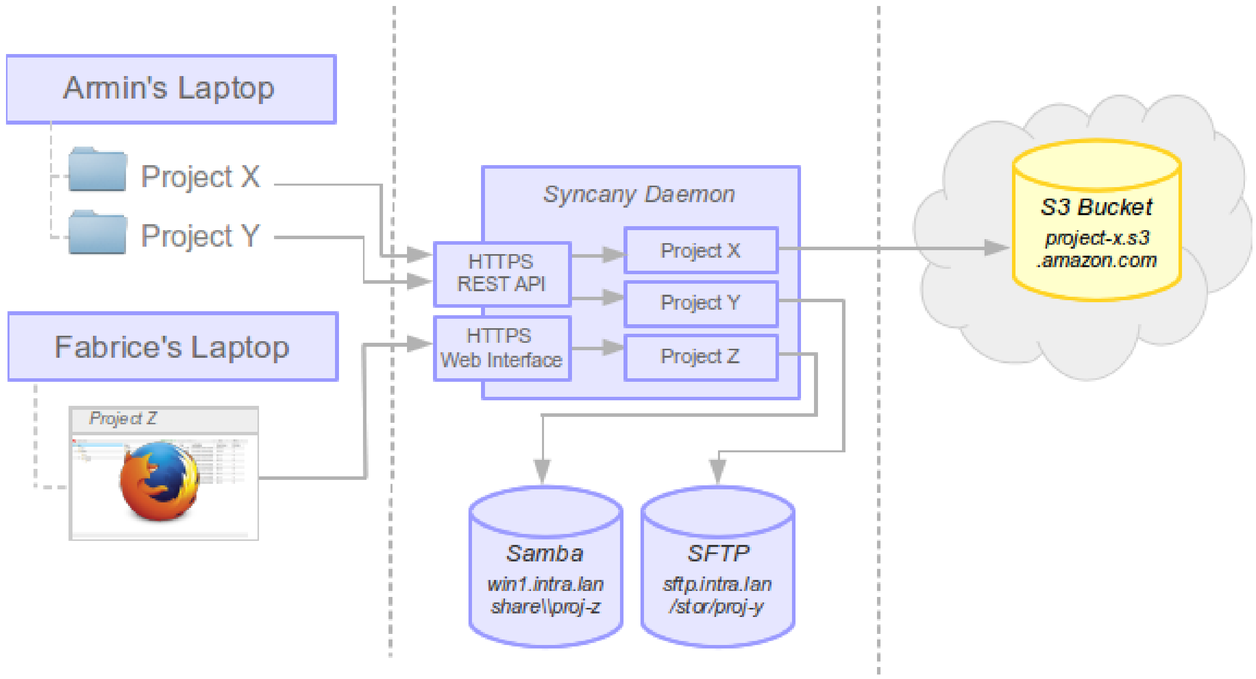
Pim sets up a Syncany shared folder with Philipp, using the shared Windows folder on his server as a backend storage. He uses the `sy init` command to do that, types in `samba` to use the Windows share plugin, and chooses a strong password to encrypt the data with. After the one-minute setup, he gets a `syncany://`-link from Syncany. This link contains all the information required to access the storage. Philipp can now use this and the password to access the Syncany folder (using the `sy connect` command). Since everything that lands in the *cloud* is encrypted with a key derived from the password, Pim and Philipp don't have to worry about a nosy storage provider or other interested parties.

The process between Philipp and Steffen is exactly the same, only that now Philipp initializes the repository on his own remote storage, and that the SFTP plugin is used.

## Use Case 2: Sharing in a company

**Note:** Note that not all of of the components required for this use case have been implemented completely. We are working on it though.

Company XYZ wants their employees to be able to share files on projects X, Y and Z. Since they have a Samba and an SFTP server lying around, they use them for projects Y and Z. The files for project X are hosted on Amazon S3 – but since the data is encrypted, company XYZ doesn't worry about their files.



Armin works on projects X and Y. He uses `sy connect` to connect to the projects repository (via the Syncany daemon). Fabrice doesn't have Syncany installed, but still needs to browse the files on project Z. He uses the web interface to do that.

CHAPTER 2

# Installation

Syncany is written in pure Java, so it available on a number of platforms. To make the installation process as easy as possible, we've pre-bundled distributables for most major operating systems (a Mac OSX bundle is still missing, see open issues). Whenever we release a new version of Syncany, we regenerate these bundles and publish them on the distribution site.

On that site, you can find **releases** and **snapshots**. Releases are built and published in a certain release cycle (currently: 2 weeks), and snapshots are built with every commit. Feel free to try out both, but be aware that snapshots are very volatile and things might break without warning.

As of today, we provide the following download possibilities:

- Windows installer (.exe)
- Ubuntu/Debian APT archive (or .deb package)
- Arch Linux AUR
- Mac OS X (.app.zip package, or via Homebrew)
- Docker Application
- Manual Installation (via .tar.gz/.zip)

**Contents**

## Installation requirements

Since Syncany heavily relies on Java, you obviously need an up-to-date version of Java installed. And that is about it. No other requirements.

- **Java/JRE >= 7**. See Oracle Website Downloads to install Java. Be sure to load the 64-bit JVM if you intend to load the 64 bit Syncany or the gui plugin will not run.
- **bash-completion >= 2** (Linux only). If it's not installed by default, it's definitely in the default packages.

## Installing the latest release

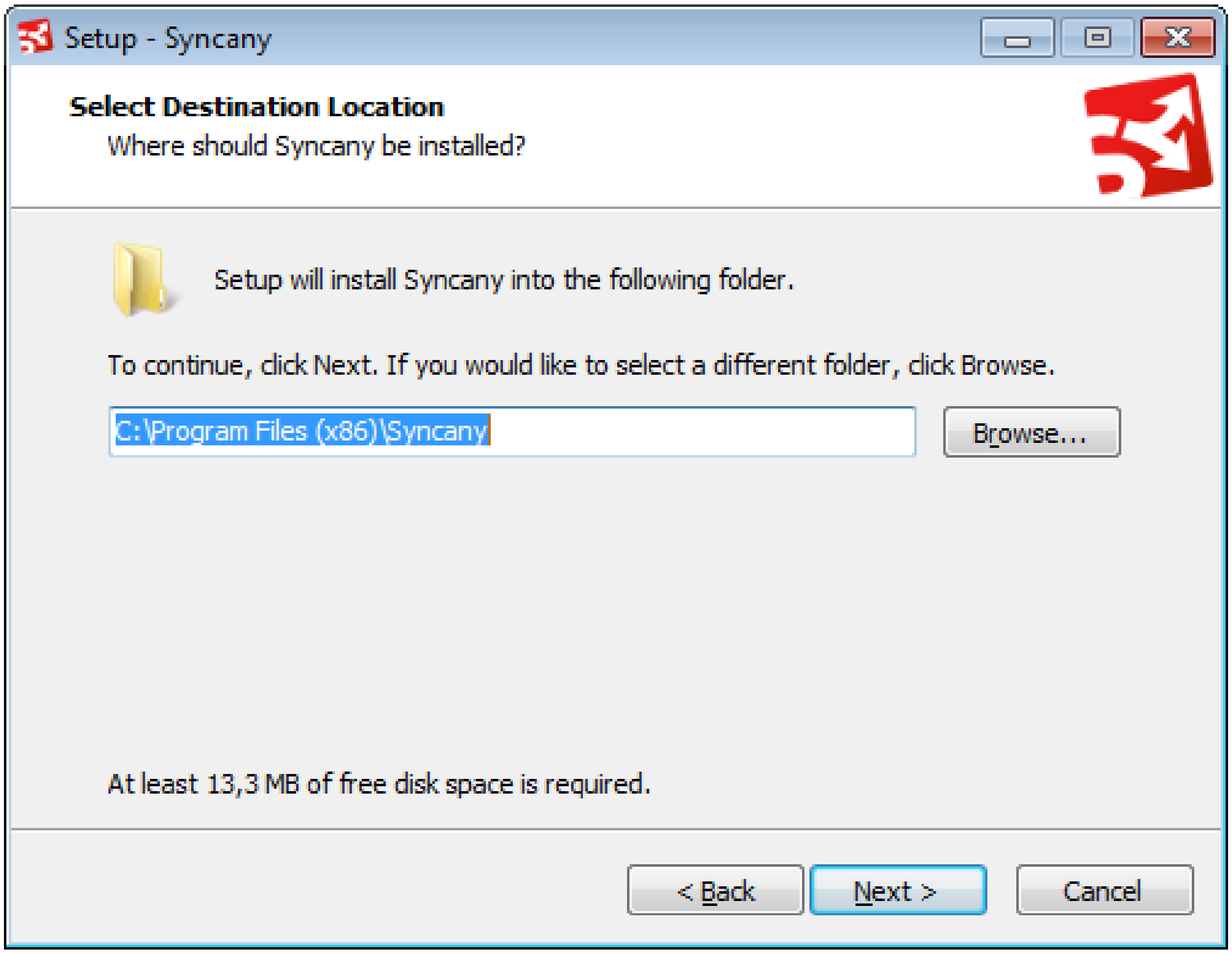
### Windows

On Windows, you can either manually extract the Syncany files from the ZIP archive, or use the installer. The latter is obviously more comfortable, but both variants have their reasons.

The installation using the Windows installer is easy and very similar to the installation of other applications.

1. Download the latest release from the distribution site (see folder *releases*). The installer files have the *.exe* ending. You can't miss it.
2. Then run the executable and follow the instructions.
3. After the installation, open the command prompt and type `sy`.

When you run the executable, you'll see a typical installer that looks something like that:


Setup - Syncany

**Select Destination Location**

Where should Syncany be installed?

Setup will install Syncany into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

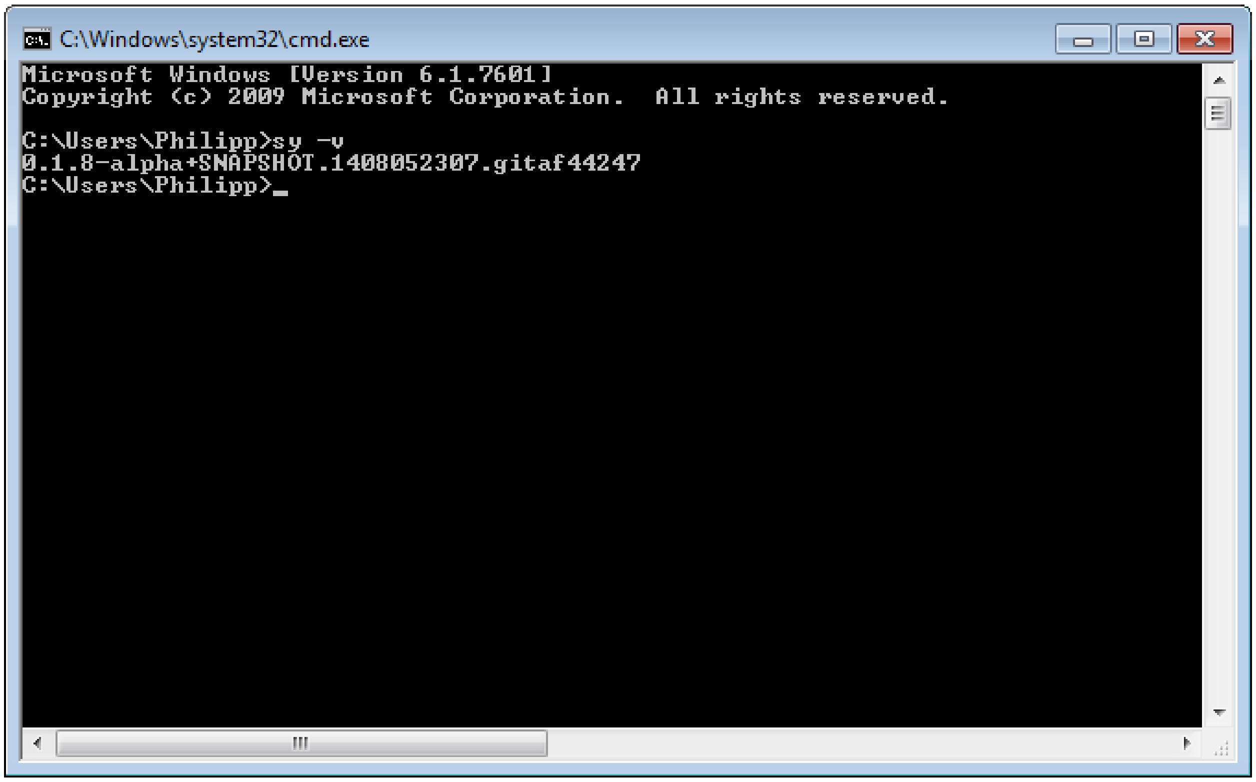
C:\Program Files (x86)\Syncany

At least 13,3 MB of free disk space is required.

Other than where to install Syncany, the installer will only give you two additional options. **If you are not sure what they mean, don't change them.**

- **Add Syncany to PATH environment variable (recommended)**: If you're unsure, leave this as is. For Syncany to be available on the command line, the command line will have to know where to look for it. If this option is unchecked, the `sy` command will not be available unless the installation path is added to the PATH environment variable.
- **Set JAVA_HOME environment variable (recommended)**: If you're unsure, leave this as is. Syncany relies on the Java Runtime Environment (JRE) and this variable tells the Syncany commands where to look for it.

After the installation is complete, open the command prompt by typing `cmd` in the Windows search box, or by navigating to *Extras*, *Command Prompt* in the menu. If everything goes well, you'll see this after typing `sy -v`:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Philipp>sy -v
0.1.8-alpha+SNAPSHOT.1408052307.gitaf44247
C:\Users\Philipp>_
```

## Debian / Ubuntu / Linux Mint

For Debian-based systems, we provide an APT archive (for installation via `apt-get`) as well as a way to manually download and install prebuilt *.deb*-packages (see distribution site).

Installing via **APT archive**:

1. Add the APT archive **http://archive.syncany.org/apt/release/** to your *additional software sources* (this might ask you to confirm the Syncany public key as a trusted key).
2. Once that is done, you can now update the package archives by `sudo apt-get update`, and install Syncany with `sudo apt-get install syncany`.
3. After the installation, open the terminal and type `sy`.

Again, for the command line lovers:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪A3002F0613D342687D70AEEE3F6B7F13651D12BD
sudo sh -c "echo deb http://archive.syncany.org/apt/release/ release main > /etc/apt/
↪sources.list.d/syncany.list"
sudo apt-get update
sudo apt-get install syncany
sy -v
```

Installing via a **.deb-package**:

1. Download the latest release Debian package from the distribution site (see folder *releases*).
2. Double-click the *.deb*-package and click "Install", or type `sudo dpkg -i syncany_<version>.deb` from the command line.

3. After the installation, open the terminal and type `sy`.

If you're more of a command line guy, simply do this:

```
wget https://syncany.org/latest.deb
sudo dpkg -i syncany-latest.deb
sy -v
```

### Arch Linux

Arch Linux users can use the `syncany` package available on the AUR to install Syncany. An AUR helper like yaourt could help with this:

```
yaourt -S syncany
```

### Mac OS X

For Mac OS X, there are currently two ways of installation:

- Download and install the the latest .app.zip package (including GUI)
- Install the Homebrew via `brew install https://get.syncany.org/homebrew/syncany.rb` (*no* GUI)

Homebrew notes: Since Syncany is still alpha software, it cannot be installed from the official Homebrew sources. Therefore we provide the needed formula on our own until we reach beta. The command above installs the most recent (pre-)release. If you want to install the bleeding edge version simply append `--HEAD` to the previous command.

If you like the Syncany daemon to start at system startup, install the provided LaunchAgent following Homebrew instructions.

### Docker

If you just want to try Syncany for a few minutes, we provide it as a containerized Docker application for Syncany. If you've installed Docker already, you can use the syncany/release repository.

```
docker pull syncany/release
docker run -ti syncany/release
syncany@e52be0b2522b:~$ sy -v
0.1.8-alpha
```

### Manual Installation (for other operating systems)

If your operating system isn't listed above, or if you just want to install Syncany manually for some other reason, simply download either the .zip or the .tar.gz archive from the distribution site (see folder *releases*). Then extract the archive and run Syncany and/or the daemon from the `bin/` folder. This is the exact same process on every major operating system:

1. Download the latest archive from the distribution site (see folder *releases*)
2. Extract the archive and run `bin/sy` to run the Syncany command line tool

If you'd like to use Syncany from there, but without having to always type the entire path to run it, you may want to place the Syncany `bin/` folder in the system's `PATH` environment variable. This will let your system know where to look for the `sy` command. It's easy to find out how to do that, but just in case:

- How to set the PATH variable on Windows
- How to set the PATH variable on Mac OSX
- How to set the PATH variable on Linux

## Installing the latest snapshot

In addition to the releases, we also provide **snapshot versions** of Syncany. Snapshots are bundles that reflect the cutting edge state of development. They are created for every single commit in our versioning system, so updates come very frequently – sometimes up to dozens of times per day. Unless you'd like to test a new feature before everyone else or you are asked by a developer to install that version, it is highly discouraged. **Things might break. They might be incompatible to previous versions. Or it might not work at all.**

If you're sure you want to install a snapshot, the process is very similar to the steps above. The only thing that differs is the file or access channel where you get the installer/package.

- For Windows, download the latest snapshot installer from the snapshots folder.
- For Debian/Ubuntu, use the Syncany APT archive.
- For Arch Linux, use the Git package on AUR.
- For Docker, use the Docker snapshot repository (`syncany/snapshot`).
- And for other operating systems, download the latest snapshot .tar.gz/.zip from the snapshots folder.

## Installing from source

Syncany hosts its code on GitHub, so if you want to compile Syncany from source, you definitely can – and we encourage you to do so. Since this is a user manual, however, we don't want to go into too much detail about the build process. There are details about building available on the Wiki page.

In short, run this:

```
git clone http://github.com/syncany/syncany
cd syncany
./gradlew installApp          (on Linux / Mac OS)
gradlew installApp            (on Windows)
```

This compiles and installs the Syncany command line client to `syncany-cli/build/install/syncany/bin/syncany`. You can run it from there.

CHAPTER 3

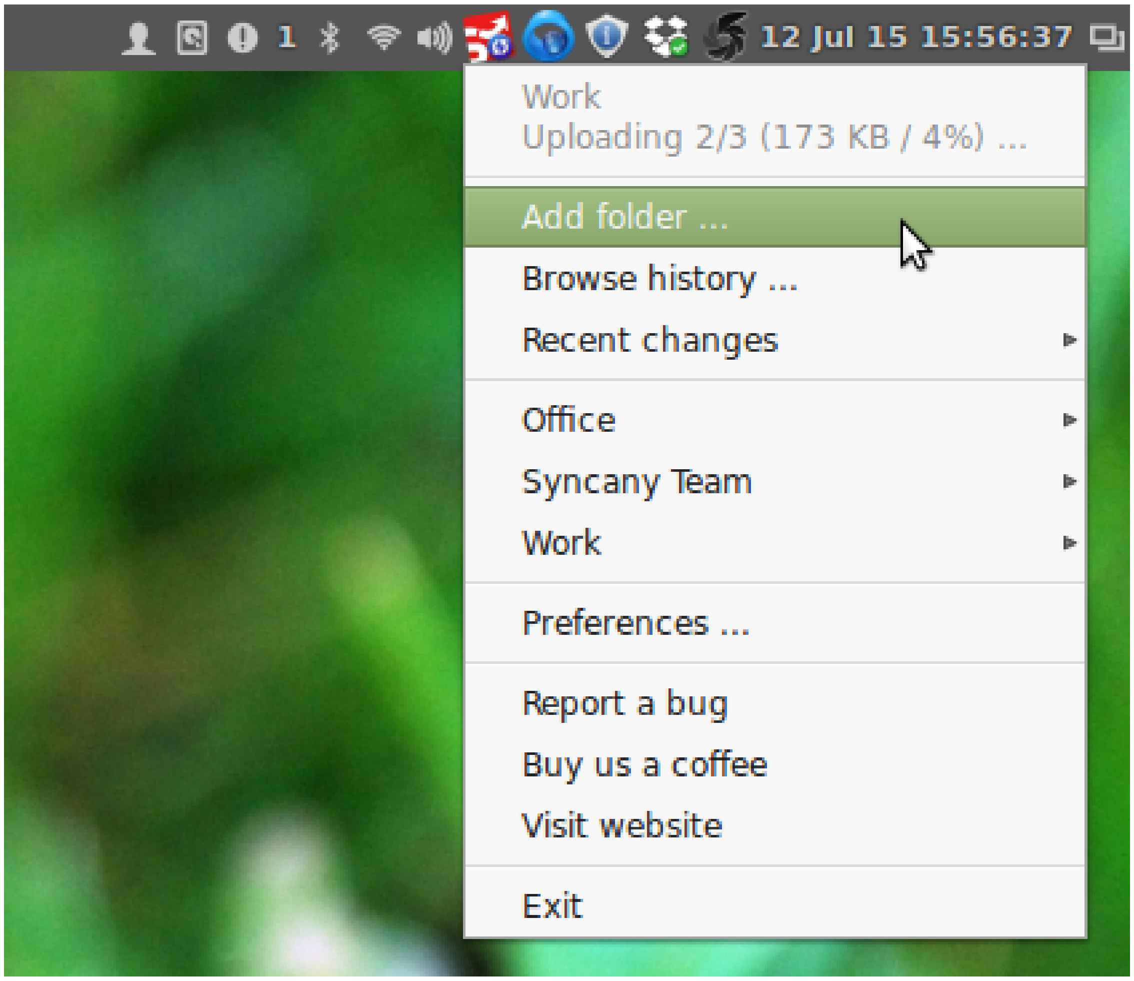
# Getting Started

Once you've *installed Syncany*, you can either use the Syncany graphical user interface (GUI), or the command line client (CLI). While the GUI is probably easier to use, the CLI offers more functionality and flexibility.

**Contents**

## Using the graphical user interface (GUI)

After installing Syncany, open it from your apps menu and if all goes well, you'll see a tray icon with the Syncany symbol in your notification area.

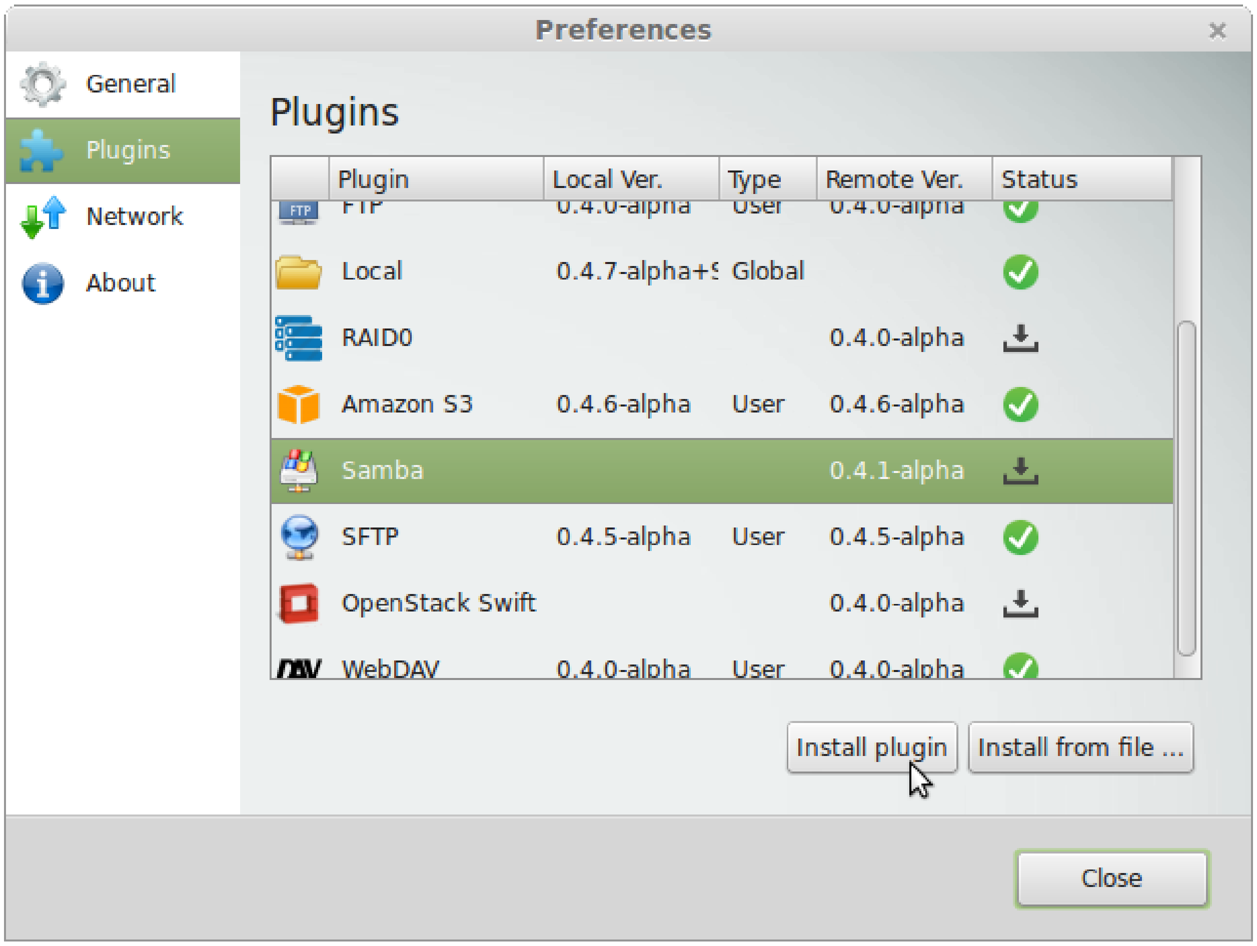


From this tray menu, you can select a new local folder to sync, change settings or install plugins. Every interaction with Syncany starts from this tray icon and its menu. Typically, if you're new to Syncany, the following steps will have to be taken:

1. Select and install a *storage plugin* via the 'Preferences/Plugins' dialog
2. Initialize a new repository or connect to an existing repository via the 'Add folder ...' wizard

## Installing a storage plugin

Syncany typically only ships with a single plugin. If you'd like to use other backend storages such as (S)FTP, Amazon S3, WebDAV and such, you'll have to install the corresponding plugin(s). To do that, open the 'Preferences ...' dialog from the tray menu and install the plugin on the plugins panel:



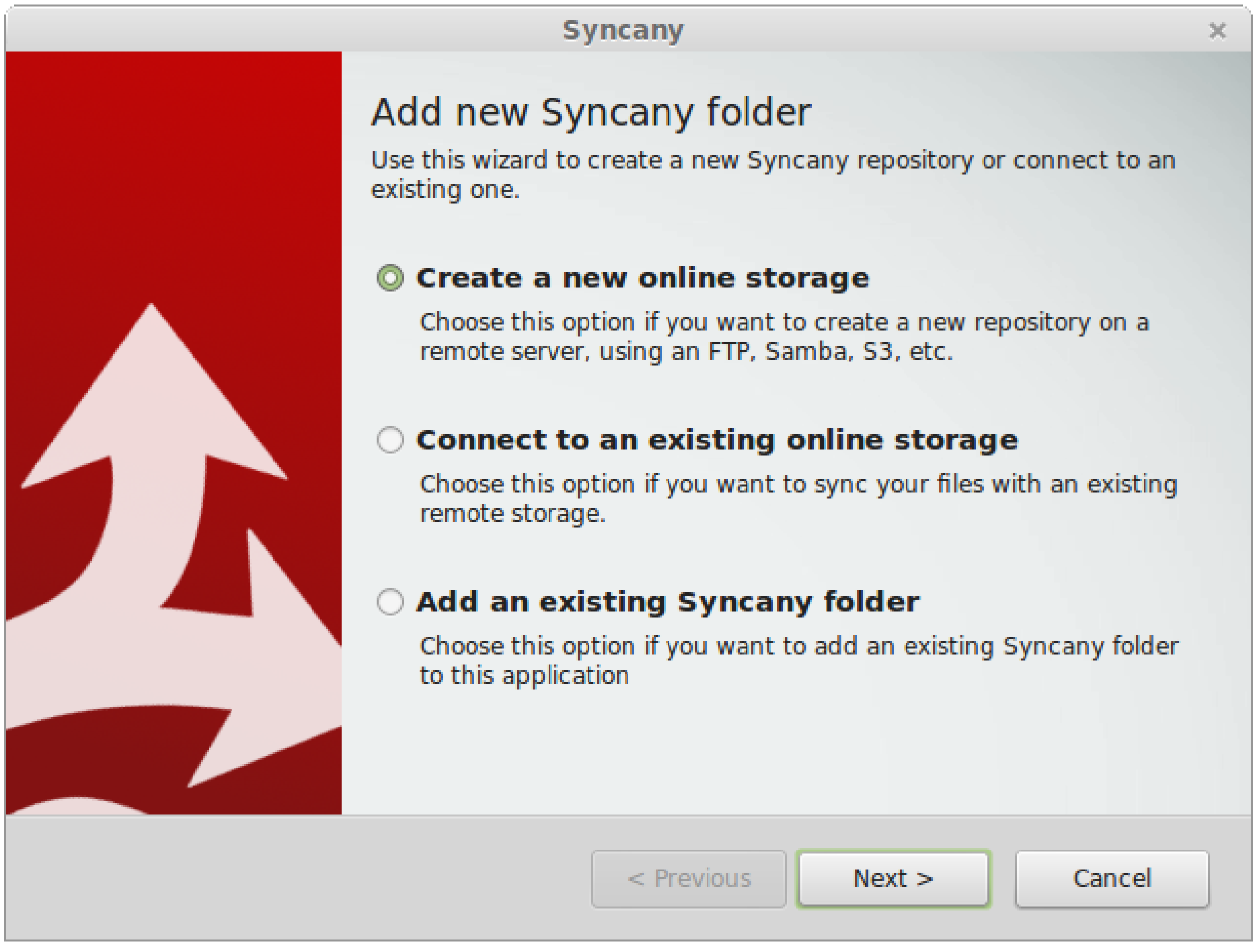
Plugins

| Plugin | Local Ver. | Type | Remote Ver. | Status |
|---|---|---|---|---|
| FTP | 0.4.0-alpha | User | 0.4.0-alpha | ✓ |
| Local | 0.4.7-alpha+ | Global | | ✓ |
| RAID0 | | | 0.4.0-alpha | ⤓ |
| Amazon S3 | 0.4.6-alpha | User | 0.4.6-alpha | ✓ |
| Samba | | | 0.4.1-alpha | ⤓ |
| SFTP | 0.4.5-alpha | User | 0.4.5-alpha | ✓ |
| OpenStack Swift | | | 0.4.0-alpha | ⤓ |
| WebDAV | 0.4.0-alpha | User | 0.4.0-alpha | ✓ |

To use the plugin, a restart of Syncany is necessary.

## Initializing a new repository

Unlike Dropbox and others, Syncany does not (and can't) set up a default sync folder. To synchronize a set of files, you need to manually add a folder to Syncany and link it to a storage backend. To do that, click on the 'Add a folder' menu item in the tray. This will bring up a wizard dialog to guide you trough the process:



Add new Syncany folder

Use this wizard to create a new Syncany repository or connect to an existing one.

[✓ Create a new online storage]

Choose this option if you want to create a new repository on a remote server, using an FTP, Samba, S3, etc.

Connect to an existing online storage

Choose this option if you want to sync your files with an existing remote storage.

Add an existing Syncany folder

Choose this option if you want to add an existing Syncany folder to this application

Among others, you will need to ...

- ... select a local folder that contains your files
- ... select a storage backend (e.g. WebDAV, Amazon S3, SFTP, ...)
- ... choose a password

Once you've gone through all of these steps, you can start adding your files to the local sync folder and connect other devices to your repository. Syncany will run as a daemon in the background and watch for changed/added/deleted files.

To see a **screencast** of the whole process, check out this YouTube video. Also check out our other *Videos*.

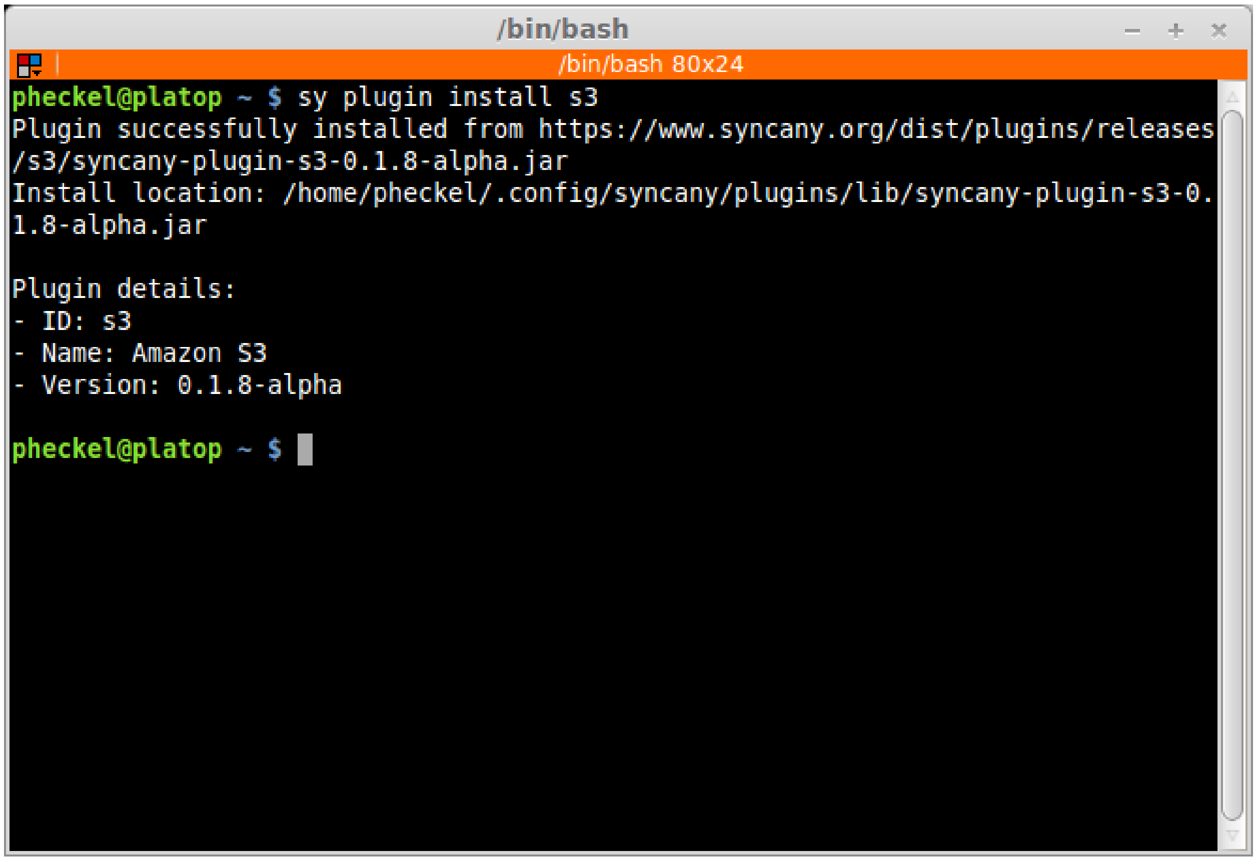
## Using the command line interface (CLI)

The command line interface is much more powerful than the Syncany GUI. To use it, you can go ahead and open a terminal window and try out the `sy` command – everything you can do with Syncany so far is implemented in this command. The typical order of commands is:

1. Install a *storage plugin* using `sy plugin install <plugin-id>`
2. Initialize a new repository with `sy init`
3. Then sync files manually with `sy up` and `sy down`, or automatically with *the daemon*
4. Connect and share with other clients with `sy connect`

## Installing a storage plugin

First choose the storage backend you'd like to use by doing `sy plugin list` and then `sy plugin install`. As of today, we've implemented a number of plugins. To get a complete list, check out the *plugins page*. For this example, we'll install the Amazon S3 plugin:

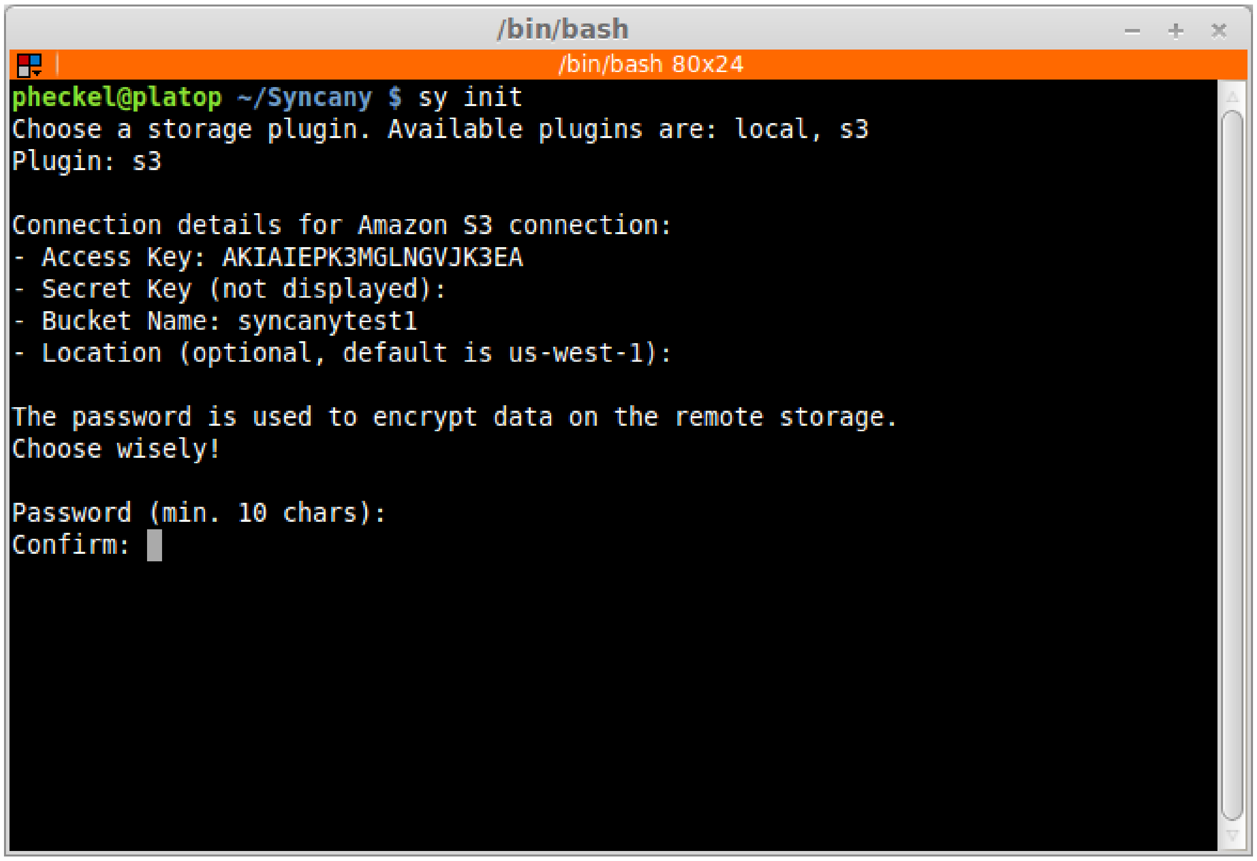


```
pheckel@platop ~ $ sy plugin install s3
Plugin successfully installed from https://www.syncany.org/dist/plugins/releases
/s3/syncany-plugin-s3-0.1.8-alpha.jar
Install location: /home/pheckel/.config/syncany/plugins/lib/syncany-plugin-s3-0.
1.8-alpha.jar

Plugin details:
- ID: s3
- Name: Amazon S3
- Version: 0.1.8-alpha

pheckel@platop ~ $ 
```

## Initializing a new repository

Once the plugin is installed (in this example the Amazon S3 plugin), you can set up a new repository: To do that, simply navigate to the folder you want to synchronize and type `sy init` command.

An **interactive command line interface** will ask you a couple of questions about how to connect to the offsite storage (hostname, credentials, etc.), and it will ask you for a password to encrypt your data with.



```
pheckel@platop ~/Syncany $ sy init
Choose a storage plugin. Available plugins are: local, s3
Plugin: s3

Connection details for Amazon S3 connection:
- Access Key: AKIAIEPK3MGLNGVJK3EA
- Secret Key (not displayed):
- Bucket Name: syncanytest1
- Location (optional, default is us-west-1):

The password is used to encrypt data on the remote storage.
Choose wisely!

Password (min. 10 chars):
Confirm:
```

As soon as you've successfully run the init command, the repository is all set up. You can now sync files to it using `sy up` and `sy down`, and you can connect other client with the `sy connect` command. The command will output a link that you can share with trusted friends or colleagues that enables them to connect to the same repository.

**Note:** Syncany will synchronize everything in the initialized folder (including sub-folders). If you are just starting out, choose a *small directory* and and get a feel for the tool before synchronizing large amounts of data. **Do not sync your home directory**!

## Syncing files

Syncany repositories can be synchronized manually via the command line, or automatically with the help of the background process (or daemon):

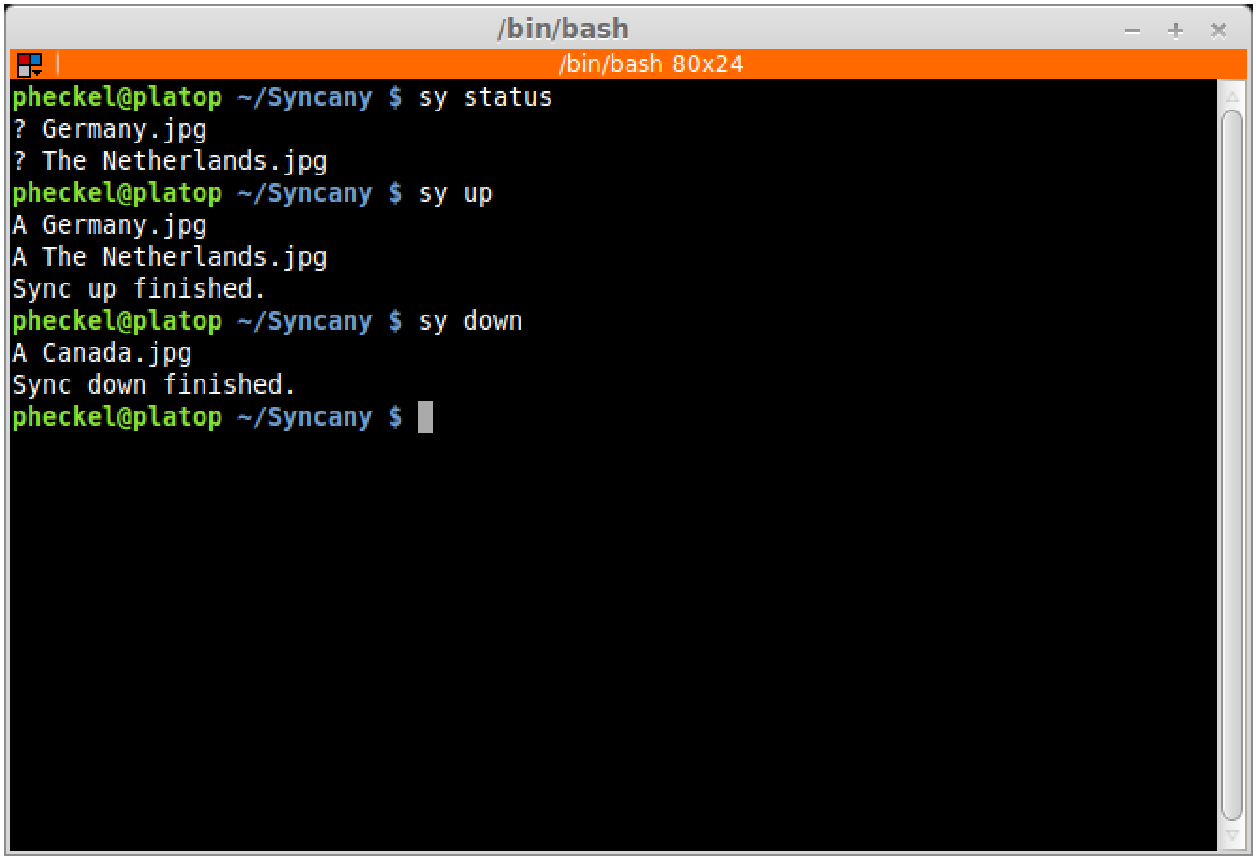
- **Syncing files manually** is useful for automated backup jobs, or if you want to use Syncany like a version control system.
- Using the **automatic syncing** turns Syncany into a Dropbox-like tool, because changes on local files are detected automatically and remote changes are synced right after they happen.

## Manually syncing files

To use Syncany for manual syncing, the `sy status`, `sy up` and `sy down` commands are your best friends. These commands work similar to a version control system:

- The `sy status` command shows you if you've made local changes

- The `sy up` command indexes and uploads these changes
- And the `sy down` command downloads and applies remote changes locally



```
pheckel@platop ~/Syncany $ sy status
? Germany.jpg
? The Netherlands.jpg
pheckel@platop ~/Syncany $ sy up
A Germany.jpg
A The Netherlands.jpg
Sync up finished.
pheckel@platop ~/Syncany $ sy down
A Canada.jpg
Sync down finished.
pheckel@platop ~/Syncany $
```

Unlike a version control system, however, there is no need to first add, then commit and then push changes – the `sy up` command combines these actions into one. Similarly, the `sy down` command fetches and applies changes in a single command.

There are a couple of other very useful commands to manage a repository and a local Syncany folder: Use `sy ls` to show the current and past file tree, `sy restore` to restore old or deleted files, and `sy cleanup` to save space on the offsite storage by removing old file versions. A full list of commands can be found in *Commands*.

## Automatically syncing files

**Note:** As of today, the automatic synchronizaton setup is not as easy as it should be. We are aware of that and are working on a solution.

To set up a Dropbox-like folder synchronizaton for a Syncany folder, the folder has to be managed by the Syncany background process (also called *the daemon*). This background process can be started with `sy daemon start`. Once the daemon is started, all registered folders are monitored for changes and remote changes are automatically applied to the local folder. All of these actions happen in the background – without the need for any intervention.

By default, calling `sy init` or `sy connect` will **not** add the added local folder to the Syncany daemon configuration. That means, that unless you call the command with the `--add-daemon` option, the folder will not be daemon-managed. If you want to use the daemon, use `sy connect --add-daemon` or `sy init --add-daemon`.

To register a folder manually or remove a folder from daemon management, the daemon can be configured using the daemon config file at `%AppData%\Syncany\daemon.xml` or `~/.config/syncany/daemon.xml`. Assuming that you'd like `/home/pim/Syncany` to be monitored and automatically synchronized, simply add the folder to the `daemon.xml` config file like this:

```
<daemon xmlns="http://syncany.org/daemon/1">
   ...
   <folders>
      <folder>
         <path>/home/pim/Syncany</path>
         <enabled>true</enabled>
      </folder>
   </folders>
</daemon>
```

To let the daemon know about the new folder, run `sy daemon restart` (or `sy daemon reload`, only on Linux).

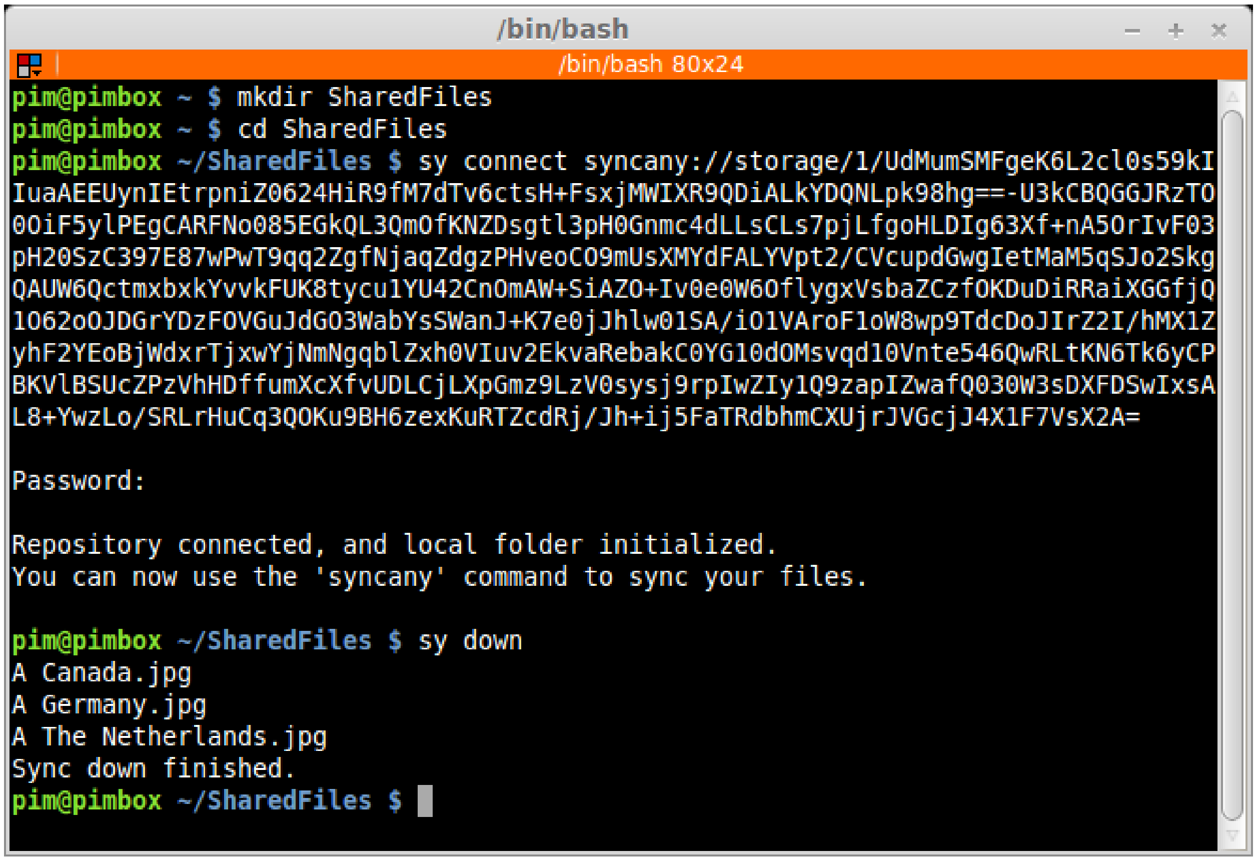
## Connecting other clients

A Syncany repository can be shared among many clients. There are two methods for new clients to connect to an existing repository:

- Use `sy connect` to manually enter the backend storage credentials (just like with `sy init`)
- Use `sy connect <syncany-link>` to avoid having to type the credentials by using a `syncany://` link

Both methods work equally well, but the `syncany://`-link method is more convenient. A `syncany://` link contains the exact same information that was initially queried by the interactive `sy init` command – namely the plugin-specific settings such as hostname, user/password, and so on. Clients that already have access to the repository can create such a link by calling `sy genlink` form within the Syncany folder.

Once a new client is connected to a repository, all of the above mentioned commands can be used to either manually or automatically sync the repository.



```
pim@pimbox ~ $ mkdir SharedFiles
pim@pimbox ~ $ cd SharedFiles
pim@pimbox ~/SharedFiles $ sy connect syncany://storage/1/UdMumSMFgeK6L2cl0s59kI
IuaAEEUynIEtrpniZ0624HiR9fM7dTv6ctsH+FsxjMWIXR9QDiALkYDQNLpk98hg==-U3kCBQGGJRzTO
0OiF5ylPEgCARFNo085EGkQL3QmOfKNZDsgtl3pH0Gnmc4dLLsCLs7pjLfgoHLDIg63Xf+nA5OrIvF03
pH20SzC397E87wPwT9qq2ZgfNjaqZdgzPHveoCO9mUsXMYdFALYVpt2/CVcupdGwgIetMaM5qSJo2Skg
QAUW6QctmxbxkYvvkFUK8tycu1YU42CnOmAW+SiAZO+Iv0e0W6OflygxVsbaZCzfOKDuDiRRaiXGGfjQ
1O62oOJDGrYDzFOVGuJdGO3WabYsSWanJ+K7e0jJhlw01SA/iO1VAroF1oW8wp9TdcDoJIrZ2I/hMX1Z
yhF2YEoBjWdxrTjxwYjNmNgqblZxh0VIuv2EkvaRebakC0YG10dOMsvqd10Vnte546QwRLtKN6Tk6yCP
BKVlBSUcZPzVhHDffumXcXfvUDLCjLXpGmz9LzV0sysj9rpIwZIy1Q9zapIZwafQ030W3sDXFDSwIxsA
L8+YwzLo/SRLrHuCq3QOKu9BH6zexKuRTZcdRj/Jh+ij5FaTRdbhmCXUjrJVGcjJ4X1F7VsX2A=

Password:

Repository connected, and local folder initialized.
You can now use the 'syncany' command to sync your files.

pim@pimbox ~/SharedFiles $ sy down
A Canada.jpg
A Germany.jpg
A The Netherlands.jpg
Sync down finished.
pim@pimbox ~/SharedFiles $
```

While the `syncany://`-link itself is encrypted and may be shared via unsecure channels, sharing the link *and* the repository password gives users read/write access to your repository and typically enables them to access the entire backend storage.

**Warning: Remember:** Do not share the `syncany://` link *and* the password with users that you do not fully trust. Users in possession of the link and the password might be able to delete/change files on the backend storage!

If, for instance, the repository is based on an FTP folder, the `syncany://` link contains the FTP username and password. Users with access to the Syncany repository can also access the FTP storage with a regular FTP client and delete/change files as they wish.

CHAPTER 4

# Concepts

There are a few very smart concepts built into Syncany that make it so flexible and differentiate it from other solutions. It's not necessary for users to understand these concepts, but it helps to understand why Syncany does things the way it does them.

**Note:** This chapter explains the basic concepts of Syncany to the interested user in a hopefully understandable way. If you just want to use Syncany and don't care how it ticks inside, **you can safely skip this chapter**.

**Contents**

## Abstraction for dumb storage through a minimal API

Syncany offers quite a bit of options when it comes to where you can store your files. As the *Plugins* page describes, the backend storage options are very flexible. The reason why that is possible is the **storage abstraction** concept of Syncany.

For a storage backend to be usable with Syncany, a plugin for that backend only has to implement a very small set of operations. This **minimal storage interface** comprises just five methods to manage files on the offsite storage, which makes it incredibly easy to implement for developers:

- `upload(local file, remote file)` - Upload a local file to the offsite storage
- `download(remote file, local file)` - Download a file from the offsite storage
- `move(remote file 1, remote file 2)` - Move file on the offsite storage
- `list(file type)` - List remote files on the offsite storage
- `delete(remote file)` - Delete file from the offsite storage

Implementing these methods (plus a few helper functions) is enough to create a plugin. Plugins don't have to deal with connection handling, synchronization aspects, security issues or notifications. In particular, plugins don't have to deal with large files or resumable uploads/downloads. Syncany uploads files with a max. size of about 4 MB and manages connections itself. Plugins don't need to be bothered with any of that - Syncany takes care of that and thereby **keeps the reponsibility of a plugin very small**.

Notably, **plugins don't have to implement any active components on the server side**. That means that there is no server component on the storage backend that has to deal with managing clients or connections. Instead, **any dumb storage backend** can be used to implement a plugin.

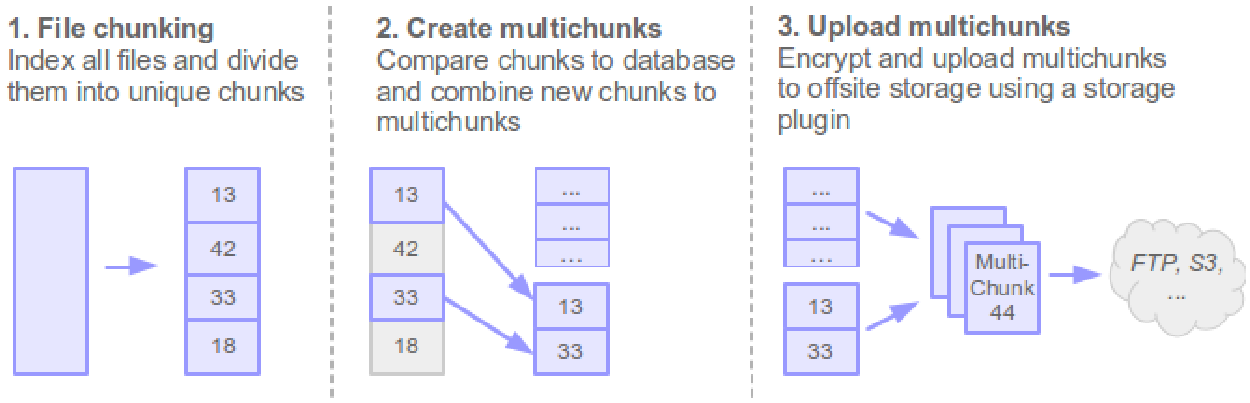
This obviously includes the classical storage systems such as (S)FTP or WebDAV, but it extends to using protocols that were't really built for storage backends: One could use IMAP to store files in e-mails, or use the Google Images/Picasa API to store files in images. With Syncany's minimal storage API, *any* storage can be used!

## Minimizing remote disk space through deduplication

Normal sync tools like rsync or Unison simply copy files from the local machine to the offsite storage. Granted, they transfer files pretty efficiently, but they store files at the offsite storage as they store them locally. Two marginally different or even identical files locally take up the same disk space as locally.

Syncany is different: Syncany uses **data deduplication** to minimize the amount of disk space used on the offsite storage. Deduplication exploits the similarities among different files to save disk space. It identifies duplicate sequences of bytes (chunks) and only stores one copy of that sequence. If a chunk appears again in the same file (or in another file of the same file set), deduplication only stores a reference to this chunk instead of its actual contents. Depending on the amount and type of input data, this technique can **significantly reduce the total amount of required storage**.[1]

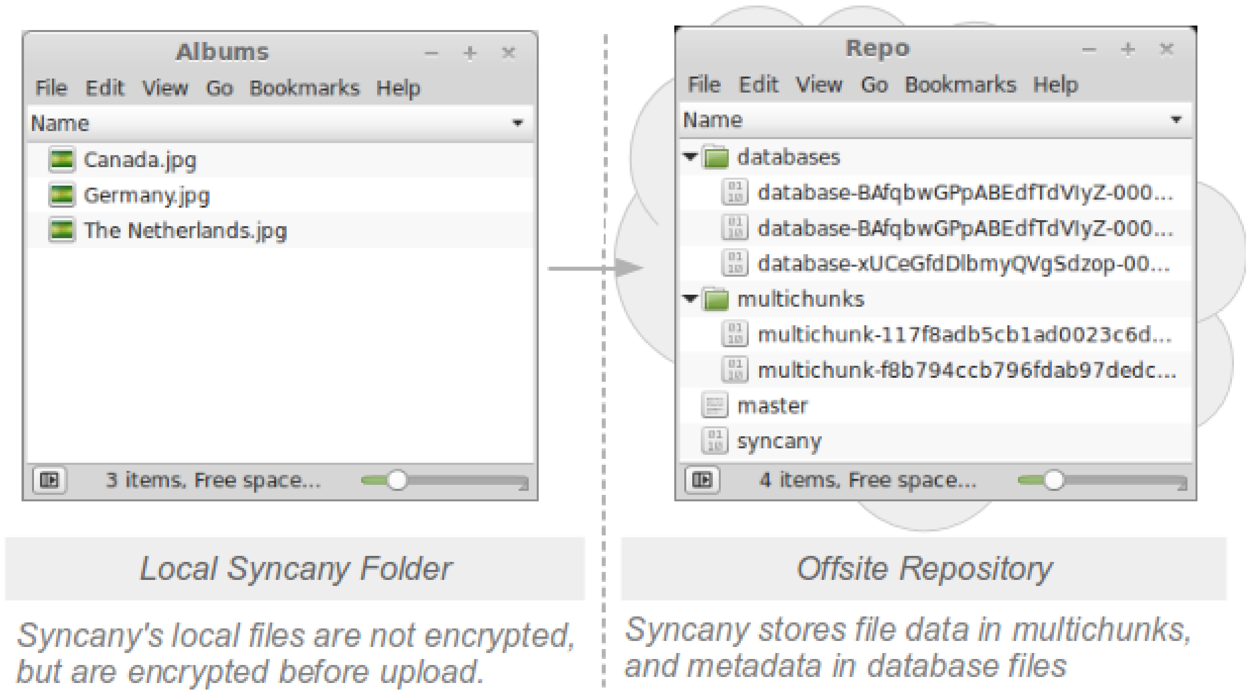
[1] Explanation of data deduplication taken from the thesis Minimizing remote storage usage and synchronization time using deduplication and multichunking: Syncany as an example.


1. File chunking
Index all files and divide them into unique chunks
13
42
33
18
2. Create multichunks
Compare chunks to database and combine new chunks to multichunks
13
42
33
18
...
...
...
13
33
3. Upload multichunks
Encrypt and upload multichunks to offsite storage using a storage plugin
...
...
...
13
33
Multi-Chunk 44
FTP, S3, ...

While this concept is typically used in backend systems of backup solutions, **Syncany performs deduplication on the client**: During the index process (in `sy up`), Syncany breaks files into chunks and compares these chunks to the database. If chunks are new, they are marked for upload. If they are already known because they appeared in other files before, they are simply referenced.

New chunks are packaged together into so-called **multichunks** (basically ZIP files containing chunks) and uploaded to the offsite storage. These multichunks exist for two reasons: Firstly, by combining chunks to multichunks, the **per-file upload latency and roundtrip time is reduced** significantly (if you've uploaded an Eclipse workspace via FTP, you know what I mean). And secondly, **multichunks are compressed and encrypted before upload** and thereby add another storage reduction mechanism and ensure data confidentiality and integrity (for encryption details see *Security*).

Multichunks only store the raw chunk data. Metadata such as as file names and permissions is stored in database files. These **database files are XML-based delta databases**, each of which represents a *commit* describing what files were added, removed or renamed. They contain information about which chunks are stored in which multichunks and how to reassemble files.


Albums
File Edit View Go Bookmarks Help
Name
Canada.jpg
Germany.jpg
The Netherlands.jpg
3 items, Free space...
Repo
File Edit View Go Bookmarks Help
Name
databases
database-BAfqbwGPpABEdfTdVlyZ-000...
database-BAfqbwGPpABEdfTdVlyZ-000...
database-xUCeGfdDlbmyQVgSdzop-00...
multichunks
multichunk-117f8adb5cb1ad0023c6d...
multichunk-f8b794ccb796fdab97dedc...
master
syncany
4 items, Free space...
Local Syncany Folder
Syncany's local files are not encrypted, but are encrypted before upload.
Offsite Repository
Syncany stores file data in multichunks, and metadata in database files

On the offsite storage, Syncany stores multichunks and database files side by side in a **repository**. Besides the multichunks and database files, this repository consists of the `master` file (containing the salt for the master key) and the `repo` file (containing repository-specific details). Since the repository is encrypted, it can only be read by another Syncany client in possession of the repository password.

## Privacy by design through client-side encryption

Unlike other sync solutions, Syncany's encryption features are built-in by design and enabled by default. In fact, not encrypting the repository doesn't bring any user advantages and only minor performance benefits. Please refer to the *Security* chapter for details.

## Trace-based synchronization through vector clocks

The synchronization algorithm is one of Syncany's core elements. Its responsibility is to detect file changes among participating workstations and to bring them to the same state.

### Trace-based synchronization

This particularly includes what is known by most file synchronizers as *update detection* and *reconciliation*[2][3][4][5][6].

**Update detection** is the process of discovering where updates have been made to the separate replicas since the last point of synchronization[2]:

- In *state-based synchronizers*[7] such as Unison or rsync, this is done by comparing the file lists of all clients. The result of the update detection process is a global file list created by merging the individual lists into one.
- In *trace-based synchronizers* such as Syncany, update detection is based on the trace log of the clients – i.e. changes of files rather than a final file list. Instead of a global file list, they generate a global file history based on the individual client histories. Trace-based synchronizers typically compare histories and detect new file versions. Update detection must additionally detect conflicting updates and determine the winner of a conflict.

Once the global file list/history has been created, the synchronizer must apply changes to the local workstation. This is done in the **reconciliation phase**. The reconciliation phase typically downloads new files, deletes old files and moves renamed files.

Due to its versioning requirements, **Syncany detects updates via trace logs** (file histories) of the individual clients. Histories of the participating clients are analyzed and compared to each other based on file identifiers, file versions, checksums and local timestamps.

### Database versions and vector clocks

Whenever a client uploads new changes by triggering `sy up`, Syncany compares the local file tree to the local database. Changes in files are packed into multichunks and described in so-called database versions (see 'database-..' files). These database versions are somewhat similar to a commit in a version control system: they describe a set of file changes. Each database version is identified by a vector clock, a logical clock that allows ordering of events in distributed systems. Using these vector clocks, Syncany knows how to order the database versions of the clients and how to resolve conflicts.

[2] Balasubramaniam and B.C. Pierce. What is a file synchronizer? In Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking, pages 98-108. ACM, 1998.

[3] Kalpana Sagar and Deepak Gupta. Remote file synchronization single-round algorithms.International Journal of Computer Applications, 4(1):32-36, July 2010. Published By Foundation of Computer Science.

[4] Bryan O'Sullivan. Distributed revision control with Mercurial, 2009.

[5] Yasushi Saito and Marc Shapiro. Optimistic replication. ACM Comput. Surv., 37:42-81, March 2005.

[6] Anne-Marie Kermarrec, Antony Rowstron, Marc Shapiro, and Peter Druschel. The icecube approach to the reconciliation of divergent replicas. In Proceedings of the twentieth annual ACM symposium on Principles of distributed computing, PODC '01, pages 210-218, New York, NY, USA, 2001. ACM.

[7] Benjamin C. Pierce and Jérôme Vouillon. What's in unison? a formal specification and reference implementation of a file synchronizer. Technical report, 2004.

Whenever a client calls `sy down`, Syncany checks for new database files (i.e. database versions) of other clients and downloads them. It extracts them and orders them using the vector clocks and then compares the local file system to the result of the changes described in the database versions of the other clients. These changes are then applied to the local file system.

### Conflict handling

When Syncany detects that two database versions were created independently of one another, i.e. their vector clocks are independent, a conflict has occurred. The conflict is resolved by simply comparing the local timestamps of the conflicting database versions to determine a winner[8]. The winner's database version(s) are applied locally and the loser's database version(s) are discarded. When the losing client detects that it lost, it'll reconcile the database versions and re-upload its changes.

## Differences and similarities to other tools

The fundamental idea of the Syncany software architecture is a mixture between a version control system like Git, SVN or Bazaar, a file synchronization software like rsync or Unison, and crypto software such as GPG.

Like in a **version control system** (VCS), Syncany keeps track of the files in a certain folder using metadata about these files (size, last modified date, checksum, etc.). It manages different versions of a file, detects if a file has been moved or changed and adds a new file version if it has. Like version control systems, Syncany knows a concept similar to a "commit", i.e. a collection of changes the local files that are uploaded to the central repository. In other ways, however, it is also very different: In contrast to Git and its friends, Syncany does not support the full range of commands that regular VCS do. For instance, there is no explicit branching or merging, no tagging and diffing. Instead, Syncany has only one trunk/master and auto-resolves conflicts when they occur (much like Dropbox does). Unlike most VCS, Syncany does not focus on text-based files, but treats all files the same (large/small, binary/text). In addition, Syncany is not limited to one or two transport protocols, but can be easily extended to many more.

The similarities to **file sync software** are quite obvious: Syncany must tackle the file synchronization problem, i.e. the problem of keeping multiple replicas of a file set in sync. Much like the widely popular rsync, Syncany compares the local files to the remote copy (or at least its metadata) using date/time/size and checksums of both whole files and parts of files, and then transfers only the changed parts to the remote location. Like rsync, Syncany tries to minimize the individual upload/download requests (and the corresponding network latency) by grouping these changes into bigger blocks. However, while rsync does that by actively gathering the file stats on the remote system, Syncany only uses the downloaded metadata, i.e. using dumb storage is possible.

Unlike any of the above mentioned tools, Syncany is **built with and around cryptography** and takes confidentiality and data integrity very seriously: Syncany generally assumes that everything but your local machine can be monitored/eavesdropped by others which is why it encrypts all data locally before uploading. As of now, Syncany only supports password-based symmetric key encryption based on configurable ciphers. By default, it uses 128 bit AES and Twofish, both in the authenticated GCM mode, but basically can support anything that Java and the Bouncy Castle crypto provider have to offer.

## Further Resources

- Master Thesis: Minimizing remote storage usage and synchronization time using deduplication and multichunking: Syncany as an example
- Syncany explained: idea, progress, development and future (part 1)

[8] The local timestamp is not used to compare which database version happened before another. It is only used as a tie-breaker to determine the winner between database versions.

- Deep into the code of Syncany - command line client, application flow and data model (part 2)

CHAPTER 5

# Commands

The Syncany command line client (CLI) comes with one single main command: `sy` (or: `syncany`). This command is the central tool to perform all of the Syncany actions, e.g. creating a new managed folder, syncing files, or restoring them. It also offers a way start and stop the Syncany user daemon, which will then run Syncany in the background.

**Contents**

## The `sy` command

The `sy` command is not just a single command, but instead offers a variety of actions to manage a Syncany folder. In general, it can be called by `sy <action> [args]`, where `action` is any of the available sub-commands and `args` is a list of optional global or sub-command specific options.

Available sub-commands:

- `init`: Initialize the current folder as a Syncany folder
- `connect`: Connect the current folder to an existing Syncany repository
- `status`: List new and changed files in local Syncany folder
- `up`: Detect local changes and upload to repository
- `ls-remote`: List changes on remote repository
- `down`: Detect remote changes and apply locally
- `watch`: Automatically synchronizes the local folder with the repo
- `cleanup`: Remove old versions from the local database and the repo
- `restore`: Restore the given file paths from the remote repository
- `genlink`: Create a syncany:// link from an existing local folder
- `ls`: Lists and filters the current and past file tree
- `plugin`: List, install and remove storage backend plugins
- `daemon`: Start and stop the background process (daemon)

Because the `init` and `connect` command initialize the current folder to a Syncany folder, they cannot be executed inside an already initialized Syncany folder. Most of the other commands behave exactly opposite to that: The commands `status`, `up`, `ls-remote`, `down`, `watch`, `cleanup`, `restore`, `genlink` and `ls` can only be execute inside an initialized Syncany folder. The only command that doesn't care where it's executed is the `plugin` command.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy / syncany*.

## `sy daemon`: Start and stop the background process (daemon)

This command manages the Syncany background process (starts, stops, etc.), also called the Syncany daemon. The command itself only offers the typical start/stop-script sub-commands, namely:

- `start`: Starts the Syncany background process (daemon), if it's not already running
- `stop`: Stop the Syncany daemon (if it is running)
- `status`: Displays whether the daemon is running and if it is, its PID
- `restart`: Restarts the daemon process by calling `stop`, then `start` (Linux only)
- `reload`: Reloads the daemon configuration (`daemon.xml`) without restarting the daemon (Linux only)
- `force-stop`: Kills the Syncany daemon; can be used if `stop` doesn't respond (Linux only)

The Syncany daemon is a user-level daemon. That means that it does not run as system service, but rather that each user may have its own daemon process running and that the daemon config is user-specific. The daemon configuration can be found at `%AppData%\Syncany\daemon.xml` (Windows) or at `~/.config/syncany/daemon.xml` (Linux).

### Example: Starting and stopping the daemon

```
$ sy daemon start
Starting daemon: .. syncanyd (pid 16336).

$ sy daemon status
Checking daemon: syncanyd running (pid 16336).

$ sy daemon stop
Stopping daemon: .. syncanyd.
```

## `sy init`: Initializing a repository

This command creates a new remote repository using the specified plugin, and initializes the local directory. Unless -I is set, the command is interactive and queries the user for input. Depending on the chosen plugin chosen (with -P or interactively), different plugin-specific options are required or optional.

Once the 'init' command was successfully executed, the initialized local folder can be synced with the newly created repository. The commands 'up', 'down', 'watch', etc. can be used. Other clients can then be connected using the 'connect' command.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy init*.

### Example 1: Create new repository (interactive mode)

```
$ sy init --create-target
Choose a storage plugin. Available plugins are: local, s3, webdav
Plugin: local

Connection details for Local connection:
- Local Folder: /tmp/x

The password is used to encrypt data on the remote storage.
Choose wisely!

Password (min. 10 chars):
Confirm:

WARNING: The password is a bit short. Less than 12 chars are not future-proof!
Are you sure you want to use it (y/n)? y

Generating master key from password (this might take a while) ...

Repository created, and local folder initialized. To share the same repository
with others, you can share this link:

   syncany://storage/1/y8aqJUCsXqPtH9KuaoAKAKO0vccIUH32k/tPRCineNLLc...

This link is encrypted with the given password, so you can safely share it.
```

```
using unsecure communication (chat, e-mail, etc.)

WARNING: The link contains the details of your repo connection which typically
         consist of usernames/password of the connection (e.g. FTP user/pass).
```

### Example 2: Create new repository (with prefilled settings)

```
$ sy init --plugin=s3 -o accessKey=AKIAJL7... -o secretKey=... \
                      -o bucket=syncanytest3 -o location=EU
...
```

## `sy connect`: Connecting to an existing repository

This command connects to an existing remote repository and initializes the local directory. The command can be called as follows:

1. Using a syncany://-link generated by either 'init' or 'genlink', the command connects to the repository given in the link. If the link is encrypted, the link/repo password must be entered.

2. If no link is given, the command acts like 'init', i.e. it queries the user for storage plugin and connection details of the repository to connect to.

Once the repository is connected, the initialized local folder can be synced with the newly created repository. The commands 'up', 'down', 'watch', etc. can be used. Other clients can then be connected using the 'connect' command.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy connect*.

### Example 1: Connect to an existing repository (with a syncany://-link)

```
sy connect syncany://storage/1/y8aqJUCsXqPtH9Ku+aoAKAKO0vcc...

Password: (user enters password)

Creating master key from password (this might take a while) ...

Repository connected, and local folder initialized.
You can now use the 'syncany' command to sync your files.
```

### Example 2: Connect to an existing repository (interactive)

```
$ sy connect
Choose a storage plugin. Available plugins are: local, s3, webdav
Plugin: local

Connection details for Local connection:
- Local Folder: /tmp/x

Password: (user enters password)

Creating master key from password (this might take a while) ...
```

```
Repository connected, and local folder initialized.
You can now use the 'syncany' command to sync your files.
```

### Example 3: Connect to an existing repository (with prefilled settings)

```
sy connect --plugin=webdav --plugin-option=url=http://dav.example.com/repo1 \
           --plugin-option=username=pheckel --plugin-option=password=<somepass>

Password: (user enters password)

Creating master key from password (this might take a while) ...

Repository connected, and local folder initialized.
You can now use the 'syncany' command to sync your files.
```

## `sy status`: Detect and display local changes

This command compares the local file tree on the disk with the local database and detects local changes. These changes are printed to the console.

Local changes are detected using the last modified date and the file size of a file. If they match the local database, the command assumes that the content has not changed (no checksum comparison). If -f is enabled, the checksum is additionally compared.

This command is used by the 'up' command to detect local changes.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy status*.

### Example 1: Basic usage (display new/changed file content)

```
$ echo "new file content" > newfile.txt
$ echo "changed content" > testfile.txt
$ sy status
? newfile.txt
M testfile.txt
```

### Example 2: Force checksum calculation per in 'status'

Forcing checksum calculation means that we don't want to rely on last modified date and size. If size and last modified date are equal, changes in local files will not be detected unless `--force-checksum` is set.

Create a file `one-thousand.txt` containing `1000`, and setting the last modified date to a specific date:

```
$ echo 1000 > one-thousand.txt
$ touch --date="Sun, 27 Apr 2014 11:11:11 +0200" one-thousand.txt
$ sy up
A one-thousand.txt
Sync up finished.
```

Now we change the `one-thousand.txt` file, but change the timestamp back to the same date as before:

```
$ echo 9999 > one-thousand.txt
$ touch --date="Sun, 27 Apr 2014 11:11:11 +0200" one-thousand.txt
```

As you can see below, the regular *sy status* command does not detect the changes. The command with the `--force-checksum` detects the changes:

```
$ sy status
No local changes.
$ sy status --force-checksum
M one-thousand.txt
```

## `sy up`: Detect local changes and upload to repository

This command detects changes in the local folder, indexes new files and uploads changes to the remote repository. If there are local changes, the command determines what has changed, packages these changes in new multichunks, and uploads them to the remote storage alongside with a delta metadata database.

To determine the local changes, the 'status' command is used. All options of the 'status' command can also be used in this command.

If there are no local changes, the 'up' command will not upload anything - no multichunks and no metadata.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy up*.

### Example 1: Basic usage (index and upload locally changed files)

```
$ sy up
A testfile.txt
A testfile2.txt
Sync up finished.
```

### Example 2: Force checksum calculation per file

Forcing checksum calculation means that we don't want to rely on last modified date and size.

```
$ sy up --force-checksum
A testfile.txt
A testfile2.txt
Sync up finished.
```

## `sy ls-remote`: Detect and display remote changes

This command detects changes in the local folder, indexes new files and uploads changes to the remote repository. If there are local changes, the command determines what has changed, packages these changes in new multichunks, and uploads them to the remote storage alongside with a delta metadata database.

To determine the local changes, the 'status' command is used. All options of the 'status' command can also be used in this command.

If there are no local changes, the 'up' command will not upload anything - no multichunks and no metadata.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy ls-remote*.

### Example: Using the ls-remote command

```
$ sy ls-remote
? db-2kjuahomsfgjmpft-0000000002
```

## `sy down`: Retrieve and apply remote changes

This command detects changes made by other clients and applies them locally. If there are remote changes, the command downloads the relevant metadata, evaluates which multichunks are required and then downloads them. It then determines what files need to be created, moved, changed or deleted, and performs these actions, if possible.

In some cases, file conflicts may occur if the local file differs from the expected file. If that happens, this command can either automatically rename conflicting files and append a filename suffix, or it can ask the user what to do.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy down*.

### Example 1: Download and apply remote changes (no conflicts)

```
$ sy down
A testfile.txt
A testfile2.txt
Sync down finished.

$ ls
testfile.txt
testfile2.txt
```

### Example 2: Download and apply remote changes (with conflicts)

```
$ echo "conflicting content" > testfile.txt
testfile.txt

$ sy down
A testfile.txt
A testfile2.txt
Sync down finished.

$ ls
testfile2.txt
testfile (pheckel's conflicted copy, 27 Apr 14, 6-46 PM).txt
testfile.txt
```

## `sy watch`: Automatically synchronizes the local folder with the repo

Automatically synchronizes the local folder with the repository. The command performs the up and down command in an interval, watches the file system for changes and subscribes to the Syncany pub/sub server.

In the default configuration (no options), the command subscribes to the Syncany pub/sub server and registers local file system watches in the locally synced folder (and all of its subfolders). When local events are registered, the command waits a few seconds (waiting for settlement) and then triggers the 'up' command. After the upload has finished, a message is published to the pub/sub server, telling other clients of this repo that there is new data. Clients subscribed to the repository's channel will receive this notification and immediately perform a 'down' command. This mechanism allows instant synchronization among clients even if a dumb storage server (such as FTP) is used.

In case file system events or pub/sub notifications are missed, the periodic synchronization using the 'down' and 'up' command is implemented as a fallback.

Note: The messages exchanged through the pub/sub server do not include any confidential data. They only include the repository identifier (randomly generated in the 'init' phase), and a client identifier (randomly generated on every restart).

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy watch*.

### Example: Automatically syncing a folder with `sy watch`

The `watch` command is a blocking command. That means when it is run, the command will not run in the background. If you desire the folder to be synced in the background, use the Syncany daemon. Details at *Automatically syncing files*.

```
$ cd ~/Syncany
$ sy watch
   (This command blocks, use the daemon if you don't want this to happen)
```

## `sy cleanup`: Remove old versions from the local database and the repo

This command performs different operations to cleanup the local database as well as the remote store. It removes old versions from the local database, deletes unused multichunks (if possible) and merges a client's own remote database files (if necessary).

Merge remote databases: Unless -M is specified, the remote databases of the local client are merged together if there are more than 15 remote databases. The purpose of this is to avoid endless amounts of small database files on the remote storage and a quicker download process for new clients.

Remove old file versions: Unless -V is specified, file versions marked as 'deleted' and files with as history longer than <count> versions will be removed from the database, and the remote storage. This will cleanup the local database and free up remote storage space. Per default, the number of available file versions per file is set to 5. This value can be overridden by setting -k.

This command uses the 'status' and 'ls-remote' commands and is only executed if there are neither local nor remote changes.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy cleanup*.

### Example 1: Default cleanup

The default cleanup command can be run manually, or triggered automatically if run in daemon mode. It'll delete old multichunks, shorten file histories and thereby free up space on the offsite storage.

```
$ sy cleanup
15 database files merged.
8 multichunk(s) deleted on remote storage (freed 12.91 MB)
19 file histories shortened.
Cleanup successful.
```

### Example 2: Shorten file histories to one version

Syncany stores multiple file versions for each file (default is 5). If `sy cleanup` is run without any options, it will still keep the last 5 versions unless `--keep-versions=N` is given, where `N` is the number of versions to keep. If you've run cleanup in the last 6 hours, you'll need to also apply `--force`.

```
$ echo version1 > file
$ sy up
A file
Sync up finished.

$ echo version2 > file
$ sy up
M file
Sync up finished.

$ sy cleanup --keep-versions=1 --force
3 database files merged.
1 multichunk(s) deleted on remote storage (freed 0.00 MB)
1 file histories shortened.
Cleanup successful.
```

## `sy restore`: Restore older versions of files from the repository

This command restores old or deleted files from the remote storage.

As long as a file is known to the local database and the corresponding chunks are available on the remote storage, it can be restored using this command. The command downloads the required chunks and assembles the file.

If no target revision is given with -r, the last version is restored. To select a revision to restore, the *sy ls* command can be used.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy restore*.

### Example 1: Restoring a previous version of a file

```
$ echo version1 > somefile
$ sy up
A somefile
Sync up finished.

```

```
$ echo version2 > somefile
$ sy up
M somefile
Sync up finished.

$ sy ls --versions somefile
14-08-23 13:11:11    rw-r--r-- --a- 9     FILE bce8ce9ce1 69101d4e4d 1 somefile
14-08-23 13:11:22    rw-r--r-- --a- 9     FILE 48a2c49dd8 69101d4e4d 2 somefile

$ sy restore --revision=1 69101d4e4d
File restored to /tmp/ssh/a/somefile (restored version 1)

$ cat "somefile (restored version 1)"
version1
```

### Example 2: Restoring a deleted file

```
$ rm somefile
$ sy up
D somefile
Sync up finished.

$ sy restore --revision=2 69101d4e4d
File restored to somefile (restored version 2)

$ cat "somefile (restored version 2)"
version2
```

## `sy ls`: Display current and historic local database

This command lists and filters the file tree based on the local database. The file tree selection can be performed using the following selection criteria:

1. Using the <path-expression>, one can select a file pattern (such as **.txt*) or sub tree (such as `subfolder/`, only with -r).

2. Using -r, the command does not only list the folder relative to the <path-expression>, but to all sub trees of it.

3. The -t option limits the result set to a certain file type ('f' for files, 'd' for directories, and 's' for symlinks). Types can be combined, e.g. `sy ls -tfs` selects files and symlinks.

4. The -D option selects the date/time at which to select the file tree, e.g. `sy ls -D20m` to select the file tree 20 minutes ago or `sy ls -D2014-05-02` to select the file tree at May 2.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy ls*.

### Example 1: Create new repository (interactive mode)

```
$ sy ls
14-07-23 22:54:07    rw-r--r--      2174138     FILE 941494aa52 3910ca5c8a 1␣
↪140628161200_IMG_3575.jpg
14-07-23 09:08:08    rwxr-xr-x         4096    FOLDER            6ba412f98b 1 Code
```

```
14-07-23 22:45:58    rwxr-xr-x        4096   FOLDER           9027a43b2b 1 Pictures
14-07-23 22:54:07    rwxr-xr-x        4096   FOLDER           08319c3f16 1 Untitled␣
↪Folder
14-07-23 21:10:05    rwxr-xr-x        4096   FOLDER           6215d124dd 1␣
↪repeatedly_compiling_test
14-07-23 22:12:11    rw-r--r--         353     FILE 3a0a1ccbba faebf2beb1 1␣
↪userconfig.xml
```

### Example 2: Listing a specific subtree (as per database)

```
$ sy ls Code/
14-07-23 09:08:08    rwxr-xr-x      4096   FOLDER              4b25720447 1 Code/fanout
```

### Example 3: Listing all directories, recursively

```
$ sy ls --recursive --types=d
14-07-23 09:08:08    rwxr-xr-x      4096   FOLDER              6ba412f98b 1 Code
14-07-23 09:08:08    rwxr-xr-x      4096   FOLDER              4b25720447 1 Code/fanout
14-07-23 09:08:08    rwxr-xr-x      4096   FOLDER              7adc2e20c5 1 Code/fanout/
↪fanout
14-07-23 09:08:08    rwxr-xr-x      4096   FOLDER              98f8df9aec 1 Code/fanout/
↪fanout/debian
14-07-23 09:08:08    rwxr-xr-x      4096   FOLDER              09fe5113f1 1 Code/fanout/
↪fanout/debian/source
14-07-23 22:45:58    rwxr-xr-x      4096   FOLDER              9027a43b2b 1 Pictures
14-07-23 22:54:07    rwxr-xr-x      4096   FOLDER              08319c3f16 1 Untitled␣
↪Folder
14-07-23 21:10:05    rwxr-xr-x      4096   FOLDER              6215d124dd 1 repeatedly_
↪compiling_test
14-07-23 21:10:05    rwxr-xr-x      4096   FOLDER              fc5a5966bb 1 repeatedly_
↪compiling_test/scriptie
```

### Example 4: Listing all versions of all PDF files in a certain folder

```
$ sy ls --versions --group repeatedly_compiling_test/scriptie/^.pdf
File 33b1042a91, repeatedly_compiling_test/scriptie/Scriptie.pdf
   14-07-23 10:28:25    rw-r--r--      273966     FILE a1d3b30444 33b1042a91 1␣
↪repeatedly_compiling_test/scriptie/Scriptie.pdf
   14-07-23 18:48:19    rw-r--r--      273966     FILE a1d3b30444 33b1042a91 2 Code/
↪repeatedly_compiling_test/scriptie/Scriptie.pdf
 * 14-07-23 21:10:05    rw-r--r--      273966     FILE a1d3b30444 33b1042a91 3␣
↪repeatedly_compiling_test/scriptie/Scriptie.pdf

File 593a67cd5e, repeatedly_compiling_test/scriptie/VoorlopigeScriptie.pdf
   14-07-23 10:28:25    rw-r--r--      247367     FILE 4b66adf265 593a67cd5e 1␣
↪repeatedly_compiling_test/scriptie/VoorlopigeScriptie.pdf
   14-07-23 18:48:19    rw-r--r--      247367     FILE 4b66adf265 593a67cd5e 2 Code/
↪repeatedly_compiling_test/scriptie/VoorlopigeScriptie.pdf
 * 14-07-23 21:10:05    rw-r--r--      247367     FILE 4b66adf265 593a67cd5e 3␣
↪repeatedly_compiling_test/scriptie/VoorlopigeScriptie.pdf
```

### Example 5: Listing all PDF files at a certain time

```
$ sy ls --date='14-07-23 18:48:20' --recursive ^.pdf
14-07-23 18:48:19    rw-r--r--      273966     FILE a1d3b30444 33b1042a91 2 Code/
↪repeatedly_compiling_test/scriptie/Scriptie.pdf
14-07-23 18:48:19    rw-r--r--      247367     FILE 4b66adf265 593a67cd5e 2 Code/
↪repeatedly_compiling_test/scriptie/VoorlopigeScriptie.pdf
```

## `sy genlink`: Create a syncany:// link from an existing local folder

This command creates a Syncany link (syncany://..) from an existing local folder. The link can then be sent to someone else to connect to the repository.

Syncany links contain the connection information of the storage backend, so in case of an FTP backend, host/user/pass/etc. would be contained in a link. If the link is shared, be aware that you are giving this information to the other users.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy genlink*.

### Example 1: Normal usage of 'genlink'

```
$ sy genlink

To share the same repository with others, you can share this link:

   syncany://storage/1/IOl4XYsdjHRazvUJCB4GPOSA+/CDhpE8ooYNkpSCSU8Bh...

This link is encrypted with the given password, so you can safely share it.
using unsecure communication (chat, e-mail, etc.)

WARNING: The link contains the details of your repo connection which typically
         consist of usernames/password of the connection (e.g. FTP user/pass).
```

### Example 2: Short output (for scripts)

```
$ sy genlink --short
syncany://storage/1/IOl4XYsdjHRazvUJCB4GPOSA+/CDhpE8o...
```

## `sy plugin`: List, install and remove storage backend plugins

This command performs three different actions: It lists the locally installed and remotely available plugins, including version information and whether plugins can be upgraded. It installs new plugins from either a given URL or a local file. It removes locally installed plugins from the user's local plugin directory.

For a detailed command reference including all command-specific options, please refer to the manual page of this command at *sy plugin*. For a detailed explanations of plugins, refer to the chapter *Plugins*.

## Example 1: List all available plugins

List only released plugins (no snapshots):

```
$ sy plugin list
Id     | Name      | Local Version | Remote Version | Inst. | Upgr.
-------+-----------+---------------+----------------+-------+------
ftp    | FTP       |               | 0.1.0-alpha    |       | yes
local  | Local     | 0.1.2-alpha   |                | yes   |
s3     | Amazon S3 |               | 0.1.0-alpha    |       | yes
sftp   | SFTP      |               | 0.1.0-alpha    |       | yes
webdav | WebDAV    |               | 0.1.0-alpha    |       | yes
```

List released plugins and snapshots:

```
$ sy plugin list --snapshots
Id     | Name      | Local Version                                | Remote Version      ␣
↪                         | Inst. | Upgr.
-------+-----------+----------------------------------------------+---------------------
↪-----------------------+-------+------
ftp    | FTP       |                                              | 0.1.0+SNAPSHOT.
↪1404181428.git1a14769       |       | yes
local  | Local     | 0.1.2-alpha+SNAPSHOT.1404200229.gitadd2848 |                     ␣
↪                         | yes   |
s3     | Amazon S3 | 0.1.0+SNAPSHOT.1404182252.git78f0f1a       | 0.1.0+SNAPSHOT.
↪1404182149.gitb7b2918       | yes   |
sftp   | SFTP      |                                              | 0.1.0-
↪alpha+SNAPSHOT.1404191549.git10ae8b7 |       | yes
webdav | WebDAV    | 0.1.0-alpha+SNAPSHOT.1404200235.git79d610f | 0.1.0-
↪alpha+SNAPSHOT.1404192343.git93fdc0b | yes   | yes
```

## Example 2: Install an available plugin

Install plugin (release version):

```
$ sy plugin install webdav
Plugin successfully installed from https://www.syncany.org/dist/plugins/releases/
↪webdav/syncany-plugin-webdav-0.1.0-alpha.jar
Install location: /home/pheckel/.config/syncany/plugins/syncany-plugin-webdav-0.1.0-
↪alpha.jar

Plugin details:
- ID: webdav
- Name: WebDAV
- Version: 0.1.0-alpha
```

Install latest snapshot of a plugin:

```
$ sy plugin install sftp --snapshot
Plugin successfully installed from https://www.syncany.org/dist/plugins/snapshots/
↪sftp/syncany-plugin-sftp-0.1.0-alpha+SNAPSHOT.1404191549.git10ae8b7.jar
Install location: /home/pheckel/.config/syncany/plugins/syncany-plugin-sftp-0.1.0-
↪alpha+SNAPSHOT.1404191549.git10ae8b7.jar

Plugin details:
- ID: sftp
- Name: SFTP
- Version: 0.1.0-alpha+SNAPSHOT.1404191549.git10ae8b7
```

### Example 3: Remove installed plugin

```
$ sy plugin remove sftp
Plugin successfully removed.
Original local was /home/pheckel/.config/syncany/plugins/syncany-plugin-sftp-0.1.0-
↪alpha+SNAPSHOT.1404191549.git10ae8b7.jar
```

CHAPTER 6

# Plugins

Plugins extend the Syncany functionality is different ways. As of today, there are two types of plugins:

- **Storage plugins** (also: connection / transfer plugins) give Syncany the ability to use a certain storage protocol or storage provider to store its repository files. Already implemented examples include FTP, SFTP, Samba or Amazon S3.
- **Web interface plugins** implement a web frontend for Syncany.

**Contents**

## About Plugins

### Installing and Removing Plugins

The `sy plugin` command offers a very easy way to list available plugins (`sy plugin list`), install new plugins (`sy plugin install`) and remove already installed plugins (`sy plugin remove`). For a detailed command reference and examples, please refer to *sy plugin: List, install and remove storage backend plugins*.

When installing new plugins or listing available plugins, the plugin command accesses the **Syncany Plugin Repository**, a (still very small) repository in which all available Syncany plugins are registered and can be downloaded from. The repository API provides information about the the newest plugins, their versions and provides a download link where to get them.

If, for instance, a user wants to install the SFTP plugin, calling `sy plugin list` will show a list of locally installed plugins, and remotely available plugins (queried through the plugin repository API):

```
$ sy plugin list
Id      | Name       | Local Version  | Remote Version  | Inst. | Upgr.
--------+------------+----------------+-----------------+-------+------
ftp     | FTP        |                | 0.1.0-alpha     |       | yes
local   | Local      | 0.1.2-alpha    |                 | yes   |
s3      | Amazon S3  |                | 0.1.0-alpha     |       | yes
sftp    | SFTP       |                | 0.1.0-alpha     |       | yes
webdav  | WebDAV     |                | 0.1.0-alpha     |       | yes
```

Installing the plugin with `sy plugin install sftp` retrieves the download link (again, via the plugin repository API), downloads the plugin JAR file, verifies its checksum and installs it locally. After that, the plugin can be used:

```
$ sy plugin install sftp
Plugin successfully installed from https://www.syncany.org/dist/plugins/releases/
↪webdav/syncany-plugin-sftp-0.1.0-alpha.jar
Install location: /home/pheckel/.config/syncany/plugins/lib/syncany-plugin-sftp-0.1.0-
↪alpha.jar
...
```

Plugins are generally installed to the user specific Syncany plugin folder. On Windows, this folder can be found at `%AppData%\Syncany\plugins\lib`, on Linux it can be found at `~/.config/syncany/plugins/lib`. Installing a plugin is nothing more than placing the plugin JAR file in that folder. If you have trouble removing a plugin, simply delete it manually from that folder.

### Plugin Configuration

Some plugins store per-user config or persist some user-specific state. The SFTP plugin, for instance, stores the known hosts file for all the known and trusted hosts. Other plugins might want to store other information.

On Windows, this per-user plugin configuration can be found at `%AppData%\Syncany\plugins\userdata\<plugin-id>\`, or at `~/.config/syncany/plugins/userdata/<plugin-id>/` on Linux. Depending on the plugin, the files in this folder may differ.

## Storage Plugins

Storage plugins are part of the core idea of Syncany: They provide a simple interface to make any type of storage usable. This is done by keeping all of the synchronization logic, file size issues and even encryption out of the plugins. Storage plugins only take care of uploading different types of files – database files, multichunk files, and so on.

Once a storage plugin is installed (see *Installing and Removing Plugins*), it can be used to create a new remote repository (`sy init`) or connect to an existing repository (`sy connect`). After you've successfully connected a local folder to a remote repository, you can synchronize files manually with `sy up` or `sy down`, or configure the daemon to automatically sync the folder in the background.

Storage plugins typically need some connection information to connect to a remote server. The FTP plugin, for instance, needs to know the hostname of the server, its port, the username/password as well as a path/folder where to store the repository. This information, the **connection settings** is stored within the managed Syncany folder in `.syncany/config.xml`.

So if your Syncany folder is at `C:\Users\Fabrice\Syncany`, you'll find the connection settings at `C:\Users\Fabrice\Syncany\.syncany\config.xml`. Depending on the type of storage plugin, the contents of this file might be different. See below for examples of the `config.xml` file.

For other users to connect to a repository, you can either provide them with these storage credentials (e.g. FTP username/password, AWS credentials, etc.), or you can share a `syncany://` link (see *sy genlink: Create a syncany:// link from an existing local folder*).

**Warning:** Users sharing a repository typically access that repository **using the same storage credentials**. Be aware that sharing a `syncany://` link and the repository password with other users also means giving away these storage credentials. **Only share a repository with people you trust with these credentials!**

### Amazon S3 Plugin

The Amazon S3 plugin (plugin identifier `s3`) uses an Amazon S3 bucket to store the Syncany repository. Amazon S3 (Simple Storage Service) is an online file storage web service offered by Amazon Web Services. It's a pretty neat pay-as-you-go service and works very well with Syncany. If you've never tried it, you can get a free account with 5 GB of storage. As of today, the plugin only supports one repository per bucket. It cannot use sub paths of a bucket as repository.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

#### Plugin Security

The plugin uses the JetS3t Amazon S3 library to access the S3 buckets. All communication is HTTPS-only, so access credentials are protected in transit. Since the actual data is encrypted before upload, data confidentiality is not an issue either.

If the Amazon S3 plugin is used, users sharing a repository typically access that repository (i.e. the S3 bucket) **using the same AWS access key and secret key**. Be aware that sharing a `syncany://` link and the repository password with other users also means giving away these storage credentials. Only share a repository with people you trust with these credentials.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **accessKey** | yes | *none* | Amazon AWS access key |
| **secretKey** | yes | *none* | Amazon AWS secret key |
| **bucket** | yes | *none* | Name of the bucket to use as repository |
| **location** | no | us-west-1 | Location of the bucket (details see below) |

The location of the bucket is any valid Amazon AWS location. As of today, valid Amazon region values are:

- Europe: `EU`, `eu-west-1`
- United States: `us-west-1`, `us-west-2`, empty string (for the US Classic Region)
- Asia/Pacific: `ap-southeast-1`, `ap-southeast-2`, `ap-northeast-1`
- Africa: `sa-east-1`

### Example `config.xml`

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="s3">
                <accessKey>AKIAIHIALEXANDREUIIE</accessKey>
                <secretKey encrypted="true">af81727a87abc68afe1428319fad...</
↪secretKey>
                <bucket>syncany-demo</bucket>
                <location>us-west-1</location>
        </connection>
</config>
```

## Dropbox Plugin

The Dropbox plugin (plugin identifier `dropbox`) uses a folder in your Dropbox as a storage backend. Data is stored in the Syncany repository format in a dedicated `Apps` folder of your Dropbox. The plugin authenticates against the Dropbox REST API via a OAuth 2.0: During `sy init`, you will be asked to navigate to the Dropbox website and copy an access token from there.

**Note:** Syncany will only use Dropbox as a storage backend, it is not an alternative Dropbox sync client. In particular, **you will not be able to read files synchronized with Syncany using your Dropbox web interface**, because Syncany files are stored in the Syncany repository format.

In addition to that, if you run both Syncany and the Dropbox client, Dropbox will **regularly show notifications** about Syncany-originated files that have been changed. Due to the fact that Dropbox cannot disable notifications for certain folders, it is not practical to run both Syncany and the Dropbox client. Instead, we suggest to only use Dropbox as a storage backend and **disable/close the Dropbox client**.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

Dropbox REST API traffic is based on HTTPS, so **tranport security is ensured**. Since Syncany itself takes care of encrypting the files before they are uploaded, the **confidentiality of your data is not at risk**. Dropbox (or any third party) cannot read your files, even if they access the encrypted files in your Dropbox folder.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **accessToken** | yes | *none* | OAuth access token displayed on the Dropbox website |
| **path** | yes | *none* | Repository folder in your Dropbox-Syncany app folder |

### Example `config.xml`

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="dropbox">
                <accessToken encrypted="true">5379020501945a9c7e6196cb2bc1...</
↪accessToken>
                <path>RepoWork</path>
        </connection>
</config>
```

## Flickr Plugin

The Flickr plugin (plugin identifier `flickr`) is a bit of a fun plugin. It encodes all the user data into PNG images and uploads them to a Flickr album. Data of all the files in a local Syncany folder is transformed into valid square PND images that can be viewed in any image view. Because Syncany repacks and encrypts data, the images appear random to the human eye.

**Note:** While Flickr offers 1 TB of storage for images, the plugin has not extensively tested for large folders. Please be aware of this!

The plugin authenticates against the Dropbox REST API via a OAuth 2.0: During `sy init`, you will be asked to navigate to the Flickr website and copy an access token from there.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

Flickr REST API traffic is based on HTTPS, so **tranport security is ensured**. Since Syncany itself takes care of encrypting the files before they are uploaded, the **confidentiality of your data is not at risk**. Flickr (or any third party) cannot read your files, even if they access the encrypted files in your Flickr albums.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **auth** | yes | *none* | Structured authentication information |
| **album** | yes | *none* | Flickr album identifier |

### Example `config.xml`

```
<config>
        ...
        <connection class="org.syncany.plugins.flickr.FlickrTransferSettings" type=
↪"flickr">
                <album>82554672157651456</album>
                <auth>
                        <token>36576...</token>
                        <tokenSecret>bd1d...</tokenSecret>
                        <user>
                                <id>585546793@N00</id>
                                <username>Philipp Roth</username>
                                <admin>false</admin>
                                <pro>false</pro>
                                <iconFarm>0</iconFarm>
                                <iconServer>0</iconServer>
                                <realName>Christian Otte</realName>
                                <photosCount>0</photosCount>
                                <bandwidthMax>0</bandwidthMax>
                                <bandwidthUsed>0</bandwidthUsed>
                                <filesizeMax>0</filesizeMax>
                                <revContact>false</revContact>
                                <revFriend>false</revFriend>
                                <revFamily>false</revFamily>
                        </user>
                        <permission>delete</permission>
                </auth>
        </connection>
</config>
```

## FTP Plugin

The FTP plugin (plugin identifier `ftp`) uses a single folder on an FTP server as repository. Since only a sub-folder is used, multiple repositories per FTP server are possible.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

As of today, the FTP plugin does not support FTPS (the TLS extension for FTP). That means that the FTP plugin **does not provide transport security** and FTP credentials might be read by an adversary (man-in-the-middle attack). Since Syncany itself takes care of encrypting the files before they are uploaded, the **confidentiality of your data is not at risk**. However, be aware that this still means that an attacker might get access to your FTP account and simply delete all of your files.

If the FTP plugin is used, users sharing a repository typically access that repository **using the same FTP username/password combination**. Be aware that sharing a `syncany://` link and the repository password with other users also means giving away these storage credentials. Only share a repository with people you trust with these credentials.

#### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **hostname** | yes | *none* | Hostname or IP address of the FTP server |
| **username** | yes | *none* | Username of the FTP user |
| **password** | yes | *none* | Password of the FTP user |
| **path** | yes | *none* | Path at which to store the repository |
| **port** | no | 21 | Port of the FTP server |

#### Example `config.xml`

```xml
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="ftp">
                <hostname>ftp.example.com</hostname>
                <username>armin</username>
                <password encrypted="true">0e2144feed0d93bc6e8d22da...</password>
                <path>/syncany/repo2</path>
                <port>21</port>
        </connection>
</config>
```

## Local Plugin

The local plugin (plugin identifier `local`) is the only built-in storage plugin. It provides a way to use a local folder as repository for Syncany. That means that instead of connecting to a remote storage and storing the repository files remotely, Syncany will use the predefined folder to store them. While that sounds quite odd at first (*why would I want to sync to a local folder?*), it actually makes quite a lot of sense for a few cases:

- **Removable devices**: If you sync or backup to a removable device, you can use the local plugin to address the target folder on that device. For instance, you'd be specifying `/mnt/backupdisk/office` or `E:\office` as a target folder.
- **Virtual file systems**: Many storage systems can already be mounted as virtual file systems. NFS, Samba, Google Drive are just a few examples. If you used a mounted folder as target, you won't even need a special Samba or NFS plugin for Syncany, because the local plugin can be used.
- **Testing**: If you want to try out Syncany or test something, the local plugin is a very simple way to do that.

### Plugin Security

Syncany assumes that the local machine is secure, so if a regular local folder (removable device or hard disk) is used, there are no security remarkds regarding this plugin. If, however, the target repository folder points to a mounted a virtual file system, it depends on the underlying protocol if/how vulnerable the system is.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **path** | yes | *none* | Local folder used to store repository files to. |

### Example `config.xml`

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="local">
                <path>/tmp/tx/c</path>
        </connection>
</config>
```

## OpenStack Swift Plugin

The Swift plugin (plugin identifier `swift`) uses an OpenStack Swift container as a storage backend. Data is stored within objects in the object container of a Swift Object Store. The plugin authenticates against the publicly available Swift API via a authentication URL, using a username and a password.

Swift uses HTTP or HTTPS as a method of transferring files to and from the remote server and authenticate users via username/password.

The HTTP and HTTPS setup are identical in terms of parameters – only the authentication URL setting differs slightly (`http://` and `https://`). However, if HTTPS is used, only server certificates signed by CAs included in the JRE/JDK will be accepted, e.g. certificates by VeriSign, GlobalSign, etc.

**Note:** At this time, this plugin **will not work with HTTPS-based backends** if the certificate is self-signed or the signed by any CA not shipped with the JRE/JDK. In particular, you will be not asked to confirm the plugin interactively/manually. This is a known issue and will hopefully be resolved soon.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

Depending on the URL configured during setup, communication is either HTTP or HTTPS. If HTTP is used, traffic between the remote server and the local machine is not encrypted – i.e. in this case, the plugin **does not provide transport security** and credentials might be read by an adversary (man-in-the-middle attack). However, since Syncany itself takes care of encrypting the files before they are uploaded, the **confidentiality of your data is not at risk**. Be aware that this still means that an attacker might get access to your account and simply delete all of your files.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **authUrl** | yes | *none* | Swift API Authentication URL (*http://* or *https://*) |
| **username** | yes | *none* | Swift username |
| **password** | yes | *none* | Swift password |

### Example `config.xml`

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="swift">
                <authUrl>https://cloud.swiftstack.com/auth/v1.0</authUrl>
                <username>sw1f7Us3r</username>
                <password encrypted="true">0e2144feed0d93bc6e8d22da...</password>
        </connection>
</config>
```

## RAID0 Plugin

The RAID0 plugin (plugin identifier `raid0`) virtually combines two storage backends into a single storage. The plugin can use any two storage plugins, e.g. an FTP folder (*FTP plugin*) and an Amazon S3 bucket (*Amazon S3 plugin*). Unlike a RAID1 (or other RAID forms), it does not mirror the storage or provide protection against the failure of one backend. It merely combines their disk space. If one of the backends fails, all repository data is lost. As of today, there is no RAID1 plugin, but we will provide it eventually.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

The RAID0 plugin uses two other storage plugins, so its security directly depends on the respective plugins. Please refer to their documentation for details.

### Options for `config.xml`

The RAID0 plugin options are a bit different from other plugins, because depending on the chosen storage plugins, the sub-options are different. If, for instance, an FTP plugin is chosen as storage 1 (`storage1:type=ftp`), the storage options are `storage1.hostname=..`, `storage1.username=..`, and so on.

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **storage1:type** | yes | *none* | Plugin identifier of the first storage backend |
| **storage1.<opt>** | yes | *none* | Plugin-specific options of first plugin |
| **storage2:type** | yes | *none* | Plugin identifier of the second storage backend |
| **storage2.<opt>** | yes | *none* | Plugin-specific options of second plugin |

### Example `config.xml`

This example uses an Amazon S3 plugin and an SFTP plugin as a backend.

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="raid0">
                <storage1 type="s3">
                        <accessKey>AKIAIHIALEXANDREUIIE</accessKey>
                        <secretKey>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</
↪secretKey>
                        <bucket>syncany-demo</bucket>
```

```
                                <location>us-west-1</location>
                        </storage1>
                        <storage2 type="sftp">
                                <hostname>example.com</hostname>
                                <username>spikeh</username>
                                <privatekey>none</privatekey>
                                <password encrypted="true">0e2144feed0d93bc6e8d22da...</
↪password>
                                <path>/home/spikeh/SyncanyRepo</path>
                                <port>22</port>
                        </storage2>
                </connection>
</config>
```

## Samba Plugin

The Samba plugin (plugin identifier `samba`) uses a single folder on a SMB/CIFS share (also known as: Windows Share) as repository. Since only a sub-folder is used, multiple repositories per SMB/CIFS server are possible.

Since Microsoft Windows comes with SMB/CIFS support out of the box, this plugin is most useful in Windows environments. Nevertheless, it works equally well with the Linux implementation Samba.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

The Samba plugin uses the jCIFS library for SMB/CIFS. Since this library only supports NT LM 0.12 (which is SMBv1), the plugin currently does not encrypt the communication to the SMB/CIFS server.

That means that the plugin **does not provide transport security** and credentials might be read by an adversary (man-in-the-middle attack). Since Syncany itself takes care of encrypting the files before they are uploaded, the **confidentiality of your data is not at risk**. However, be aware that this still means that an attacker might get access to your SMB/CIFS account and simply delete all of your files.

If the Samba plugin is used, users sharing a repository typically access that repository **using the same username/password combination**. Be aware that sharing a `syncany://` link and the repository password with other users also means giving away these storage credentials. Only share a repository with people you trust with these credentials.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **hostname** | yes | *none* | Hostname or IP address of the Samba server |
| **username** | yes | *none* | Username of the Samba user |
| **password** | yes | *none* | Password of the samba user |
| **share** | yes | *none* | Name of the Samba share |
| **path** | no | / | Sub path of the Samba share |

### Example `config.xml`

This example uses the folder `Repo1` on the `Repositories` share for storing the files. The UNC path for this would be: `\\192.168.1.25\Repositories\Repo1`.

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="samba">
                <hostname>192.168.1.25</hostname>
                <username>Philipp</username>
                <password encrypted="true">0e99b946577d26376c64b59a...</password>
                <share>Repositories</share>
                <path>Repo1</path>
        </connection>
</config>
```

## SFTP Plugin

The SFTP plugin (plugin identifier `sftp`) uses a single folder on an SSH/SFTP server as repository. Since only a sub-folder is used, multiple repositories per SFTP server are possible. The plugin supports username/password-based authentication as well as public key based authentication:

- **Password-based authentication:** To use the password-based auth mechanism, a valid SSH user must exist. Initializing a new repository (or connecting to an existing one) is pretty straight forward: Just enter username and password, leave public key related properties empty, and you're good.
- **Public key based authentication:** To authenticate at the SSH/SFTP server using public key authentication, the public key of the local machine must be present in the remote server's authorized keys (use `ssh-copy-id` to copy over your public key). If that is the case, the `password` setting is interpreted as the private key's password.

If public key authentication is used, the first time you'll connect to a server, you'll be asked to verify the authenticity of the key fingerprint. If you have verified the key, Syncany will store the key at `~/.config/syncany/plugins/userdata/sftp/known_hosts` (Linux) or `%AppData%\Syncany\plugins\userdata\sftp\known_hosts` (Windows):

```
SSH/SFTP Confirmation
---------------------
The authenticity of host 'example.com' can't be established.
RSA key fingerprint is b0:48:b7:9d:a5:56:a6:e5:5a:49:94:29:5e:73:e4:95.
Are you sure you want to continue connecting?
```

Note that if public key authentication is used, `syncany://` links **will not work**, because the private key isn't (and should not be) part of the link itself. Syncany will generate a link, but it won't work, unless the the public key of the other user/machine is available at the same path and was also copied to the authorized keys at the SSH/SFTP server.

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

The plugin uses the JSch Java Secure Channel library. All communication is SSH/SFTP-baed, so access credentials are protected in transit. Since the actual data is encrypted before upload, data confidentiality is not an issue either.

If the SFTP plugin is used, users sharing a repository typically access that repository **using the same SFTP username/password combination** (unless public key authentication is used). Be aware that sharing a `syncany://` link and the repository password with other users also means giving away these storage credentials. Only share a repository with people you trust with these credentials.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **hostname** | yes | *none* | Hostname or IP address of the SFTP server |
| **username** | yes | *none* | Username of the SFTP user |
| **privatekey** | yes | "none" | Private key path (if public key auth is used) |
| **password** | yes | *none* | Password of the SFTP user or priv. key password |
| **path** | yes | *none* | Path at which to store the repository |
| **port** | no | 22 | Port of the FTP server |

**Please note:** If `privatekey` is set to `"none"`, the `password` is interepreted as the `username``s password. If ``privatekey` is set, the `password` is interpreted as the password of the private key. If the private key is not password protected, leave the password empty.

### Example `config.xml`

#### With username/password

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="sftp">
                <hostname>example.com</hostname>
                <username>spikeh</username>
                <privatekey>none</privatekey>
                <!-- User password -->
                <password encrypted="true">0e2144feed0d93bc6e8d22da...</password>
                <path>/home/spikeh/SyncanyRepo</path>
                <port>22</port>
        </connection>
</config>
```

#### With private key authentication

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="sftp">
                <hostname>ftp.example.com</hostname>
                <username>armin</username>
                <privatekey>/home/localuser/.ssh/id_rsa</privatekey>
                <!-- Private key password -->
                <password encrypted="true">0e2144feed0d93bc6e8d22da...</password>
                <path>/home/spikeh/SyncanyRepo</path>
                <port>22</port>
        </connection>
</config>
```

## WebDAV Plugin

The WebDAV plugin (plugin identifier `webdav`) uses a single folder on a WebDAV server as repository. Since only a sub-folder is used, multiple repositories per WebDAV server are possible. The plugin supports HTTP and HTTPS connections and authenticates users via username/password.

The HTTP and HTTPS setup are identical in terms of parameters – only the URL setting differs slightly (`http://` and `https://`). However, if HTTPS is used, the first time you connect to the server (during `sy init` or `sy connect`), Syncany will ask you to confirm the server certificate. This will happen for all certificates (even if they are signed by one of the large CAs):

```
Unknown SSL/TLS certificate
---------------------------
Owner: CN=*.syncany.org, OU=Domain Control Validated
Issuer: CN=GlobalSign Domain Validation CA - SHA256 - G2, O=GlobalSign nv-sa, C=BE
Serial number: 1492271418628120790652059091142976109636803
Valid from Mon Apr 14 23:01:38 CEST 2014 until: Wed Apr 15 23:01:38 CEST 2015
Certificate fingerprints:
 MD5:   60:FB:F7:F1:E1:9E:D6:74:06:41:03:01:16:D6:19:D3
 SHA1: DC:A8:5F:FA:1D:9D:92:A7:1C:8E:22:C6:43:9B:96:9E:62:13:C7:25
 SHA256: 84:DF:92:99:86:15:AF:A6:8D:EC:74:5C:13:BE:18:75:BC:08:34:...

Do you want to trust this certificate? (y/n)?
```

Once you've accepted this certificate, it is added to the *user-specific trust store* at `~/.config/syncany/truststore.jks` (Linux) or `%AppData\Syncany\truststore.jks` (Windows).

The plugin is not installed by default, but it can be easily installed using the `sy plugin install` command. For details about how to use this command, refer to the command reference at *sy plugin: List, install and remove storage backend plugins*.

### Plugin Security

The WebDAV plugin uses the Sardine WebDAV library. Depending on the URL configured during setup, communication is either HTTP or HTTPS.

If HTTP is used, traffic between the remote server and the local machine is not encrypted – i.e. in this case, the plugin **does not provide transport security** and WebDAV credentials might be read by an adversary (man-in-the-middle attack). However, since Syncany itself takes care of encrypting the files before they are uploaded, the **confidentiality of your data is not at risk**. Be aware that this still means that an attacker might get access to your WebDAV account and simply delete all of your files.

### Options for `config.xml`

| Plugin Option | Mandatory | Default Value | Description |
|---|---|---|---|
| **url** | yes | *none* | Hostname or IP address of the WebDAV server |
| **username** | yes | *none* | Username of the WebDAV user (basic auth) |
| **password** | yes | *none* | Password of the WebDAV user (basic auth) |

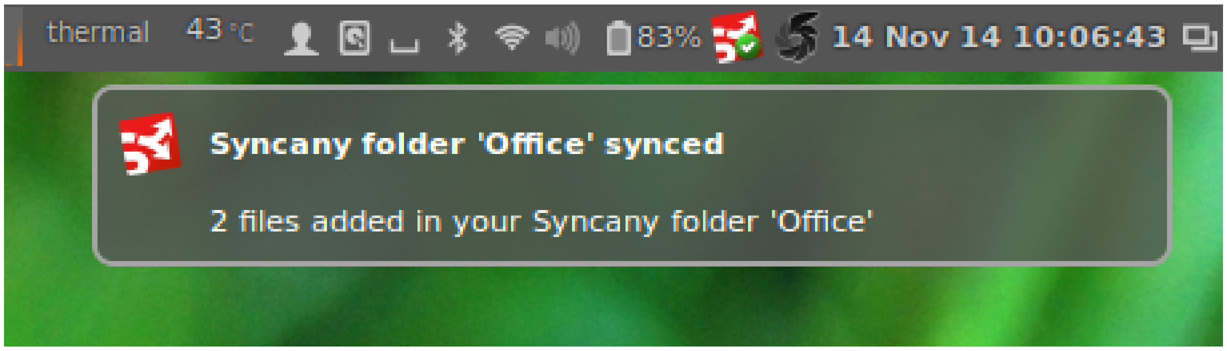
### Example `config.xml`

```
<config xmlns="http://syncany.org/config/1">
        ...
        <connection type="webdav">
```

```
                <url>https://dav.example.com:8080/dav/repo1</url>
                <username>christof</username>
                <password encrypted="true">0e99b946577d26376c64b59a...</password>
        </connection>
</config>
```

## Graphical User Interface Plugin

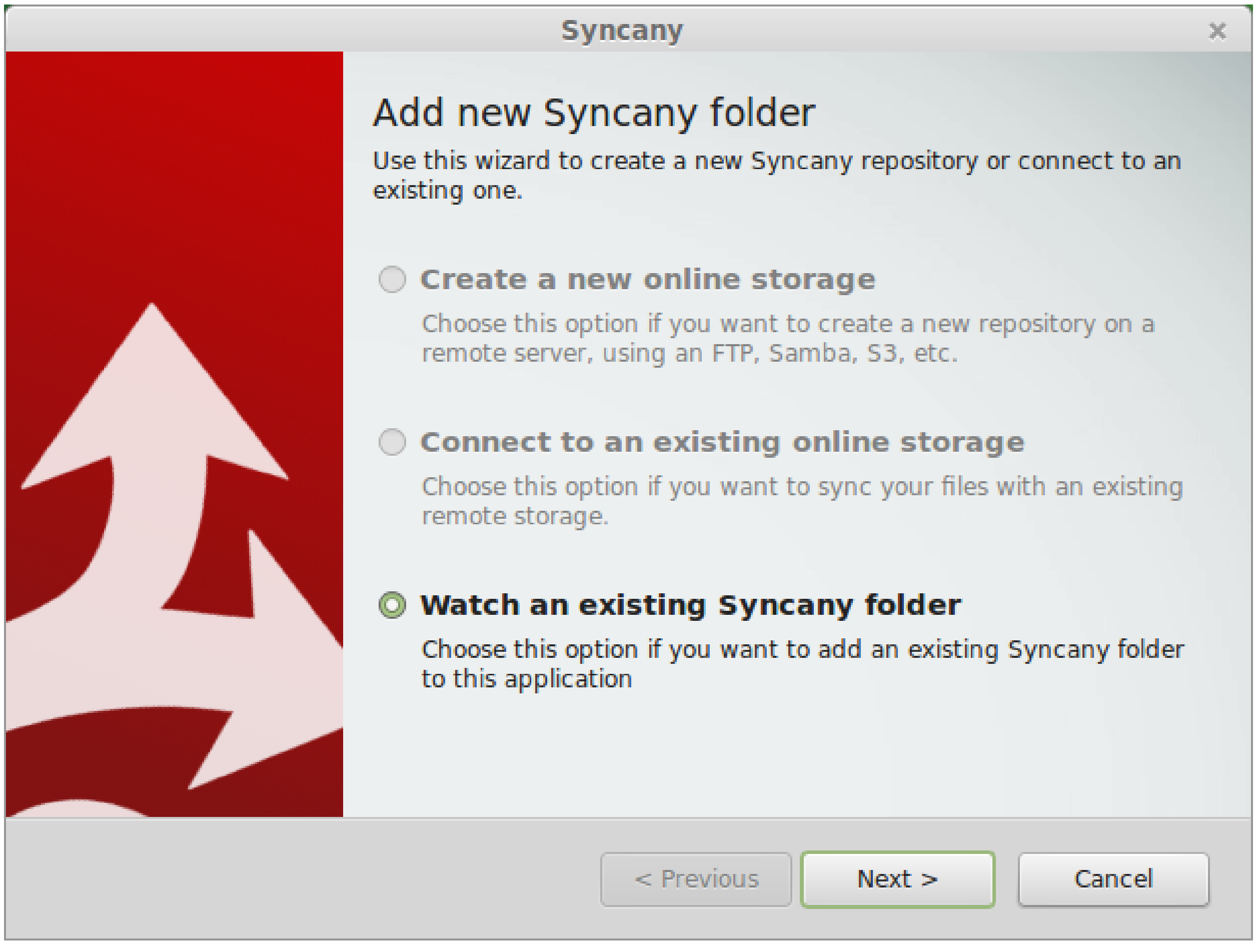
The GUI plugin (plugin identifier `gui`) provides a graphical user interface for Syncany. Since Syncany mainly runs in the background, the user interface is very minimal. It consists of a tray icon and a small wizard to initialize or connect Syncany folders. It connects to the background daemon and displays all daemon-managed Syncany folders. Folders can either be added via the GUI, or by adding it to the `daemon.xml` (see *Daemon Config (daemon.xml)*).



Syncany folder 'Office' synced

2 files added in your Syncany folder 'Office'

The tray icon indicates the progress of all daemon-managed Syncany folders – whether they are syncing or in-sync as well as a more detailed status information in form of a status text and popup notifications.


Syncany

Add new Syncany folder

Use this wizard to create a new Syncany repository or connect to an existing one.

Create a new online storage

Choose this option if you want to create a new repository on a remote server, using an FTP, Samba, S3, etc.

Connect to an existing online storage

Choose this option if you want to sync your files with an existing remote storage.

[✓ Watch an existing Syncany folder]

Choose this option if you want to add an existing Syncany folder to this application

Like other plugins, the GUI can be installed via `sy plugin install` and then run from the command line using `sy gui`. Even though this works, the GUI plugin is not meant to be installed and run like this. Instead it should be installed using the respective installers or packages. That way, Syncany can be started from the menu. For Debian/Ubuntu, a .deb package is offered; for Windows, a separate installer is provided.

## Web Interface Plugins

Web Interface plugins are a way to provide a web frontend to Syncany folders managed by a Syncany daemon. If a web interface plugin is installed, a web based frontend will be available via the web browser. Like any other plugin, web interface plugins can be installed with `sy plugin install` and are available after restarting the Syncany daemon (see details about the plugin installation at *sy plugin: List, install and remove storage backend plugins*).

In the default configuration, the web interface is served by the internal web server at port 8443 and can be accessed at https://localhost:8443. The web server settings can be changed by modifying the `daemon.xml` file as described at *Daemon Config (daemon.xml)*.

### Simple Web Interface Plugin

**Note:** The Simple Web Interface plugin is a proof-of-concept implementation. It is available as a snapshot, but it **is currently NOT functional**. We are still looking for a web frontend developer to take over / rewrite the web frontend.

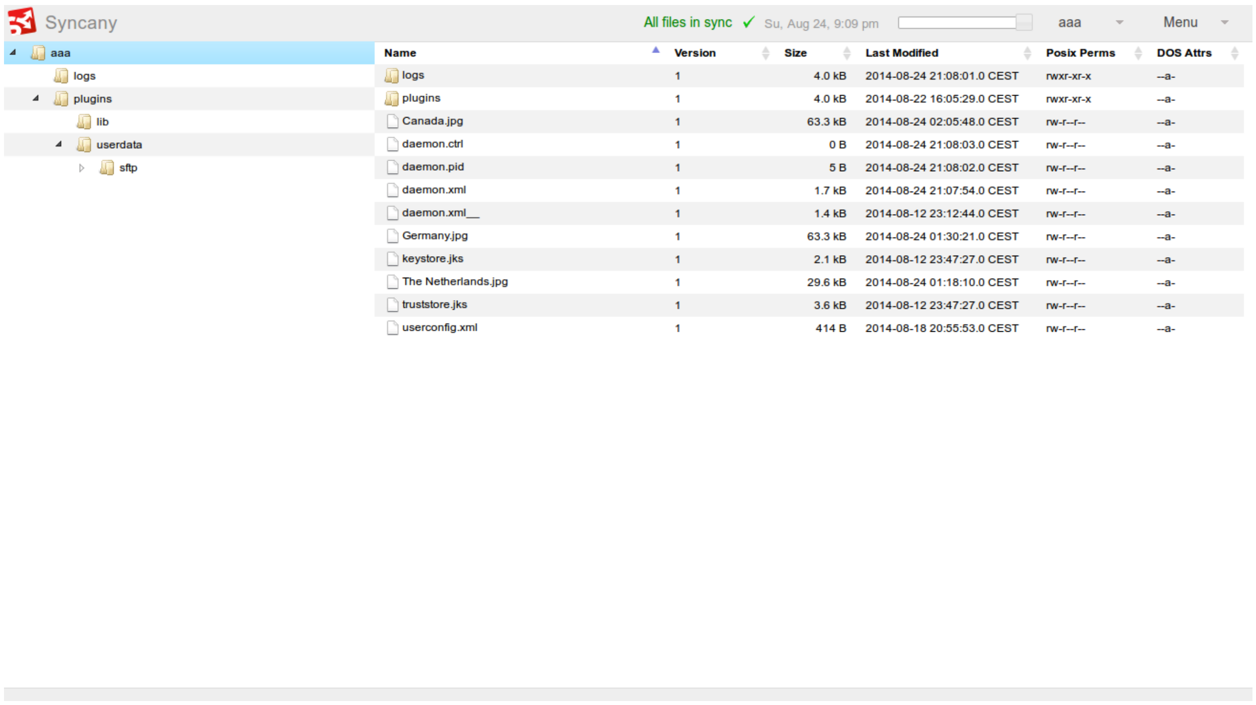
Please refer to the corresponding GitHub issue.

The Simple Web Interface plugin (plugin identifier `simpleweb`) provides access to the daemon-managed Syncany folders, i.e. all folders configured in the `daemon.xml` (see *Daemon Config (daemon.xml)*). The web frontend currently implements the following functionalities:

- Display the file tree at different times (current and past)
- Display file history of a file (old versions)
- Restore old versions of a file
- Download a file (current or past version)

To install the plugin, use `sy plugin install simpleweb --snapshot`. Make sure to enable the `--snapshot` flag, because there is no official release of the plugin (yet).

As of today, the web interface looks like this:



| Name | Version | Size | Last Modified | Posix Perms | DOS Attrs |
|---|---|---|---|---|---|
| logs | 1 | 4.0 kB | 2014-08-24 21:08:01.0 CEST | rwxr-xr-x | --a- |
| plugins | 1 | 4.0 kB | 2014-08-22 16:05:29.0 CEST | rwxr-xr-x | --a- |
| Canada.jpg | 1 | 63.3 kB | 2014-08-24 02:05:48.0 CEST | rw-r--r-- | --a- |
| daemon.ctrl | 1 | 0 B | 2014-08-24 21:08:03.0 CEST | rw-r--r-- | --a- |
| daemon.pid | 1 | 5 B | 2014-08-24 21:08:02.0 CEST | rw-r--r-- | --a- |
| daemon.xml | 1 | 1.7 kB | 2014-08-24 21:07:54.0 CEST | rw-r--r-- | --a- |
| daemon.xml__ | 1 | 1.4 kB | 2014-08-12 23:12:44.0 CEST | rw-r--r-- | --a- |
| Germany.jpg | 1 | 63.3 kB | 2014-08-24 01:30:21.0 CEST | rw-r--r-- | --a- |
| keystore.jks | 1 | 2.1 kB | 2014-08-12 23:47:27.0 CEST | rw-r--r-- | --a- |
| The Netherlands.jpg | 1 | 29.6 kB | 2014-08-24 01:18:10.0 CEST | rw-r--r-- | --a- |
| truststore.jks | 1 | 3.6 kB | 2014-08-12 23:47:27.0 CEST | rw-r--r-- | --a- |
| userconfig.xml | 1 | 414 B | 2014-08-18 20:55:53.0 CEST | rw-r--r-- | --a- |

CHAPTER 7

# Security

Syncany takes security very seriously. Our goal is to protect the confidentiality of user data stored on remote servers – meaning that whatever leaves the local computer is encrypted. This chapter explains the **security concepts** of Syncany to the interested user in a hopefully understandable way. Due to the complexity of the subject, however, this is a rather technical chapter. If you just want to use Syncany, **you can safely skip this chapter**.

**Contents**

## Prologue

Although we take special care designing and implementing the security-related parts of Syncany, we are not cryptographers. We have asked multiple parties to peer review our concept and implementation, and we hereby do that again: **Please review our security concepts and implementation!** If you believe that any of the concepts and/or the implementation is incorrect, insecure or can be attacked, please let us know by creating a new issue or by contacting us directly for major issues.

## Security Assumptions

Syncany's central security goal is to protect the user's data against attacks on **confidentiality** and **integrity**. Syncany provides data confidentiality and detects attacks on the integrity of data, but does not provide mechanisms to protect against attacks on the availability of data.

1. Syncany assumes the **storage provider and network infrastructure cannot be trusted**.
2. In contrast to that, Syncany assumes that the **local machine is secure**.
3. Users sharing a repository **trust each other completely**.

## Security Concepts

There are two security/cryptography-related topics in Syncany:

1. **Pre-upload encryption**: Before uploading anything to the offsite storage, Syncany encrypts files locally. This is done for metadata (database files) and for the actual data (multichunk files). The ciphers and modes as well as the resulting crypto file format is described in *Encryption Scheme*.
2. **Auth-only and HTTPS-only daemon/server**: Accessing the daemon API or web interface is HTTPS-only using either self-signed or your own certificates; and allows only authenticated users. Details are described in *HTTPS-only Daemon*.

### Encryption Scheme

For users to share a repository with Syncany, they must trust each other completely: They share the pre-shared **repository password** to access the Syncany repository and use the same credentials to the offsite storage.

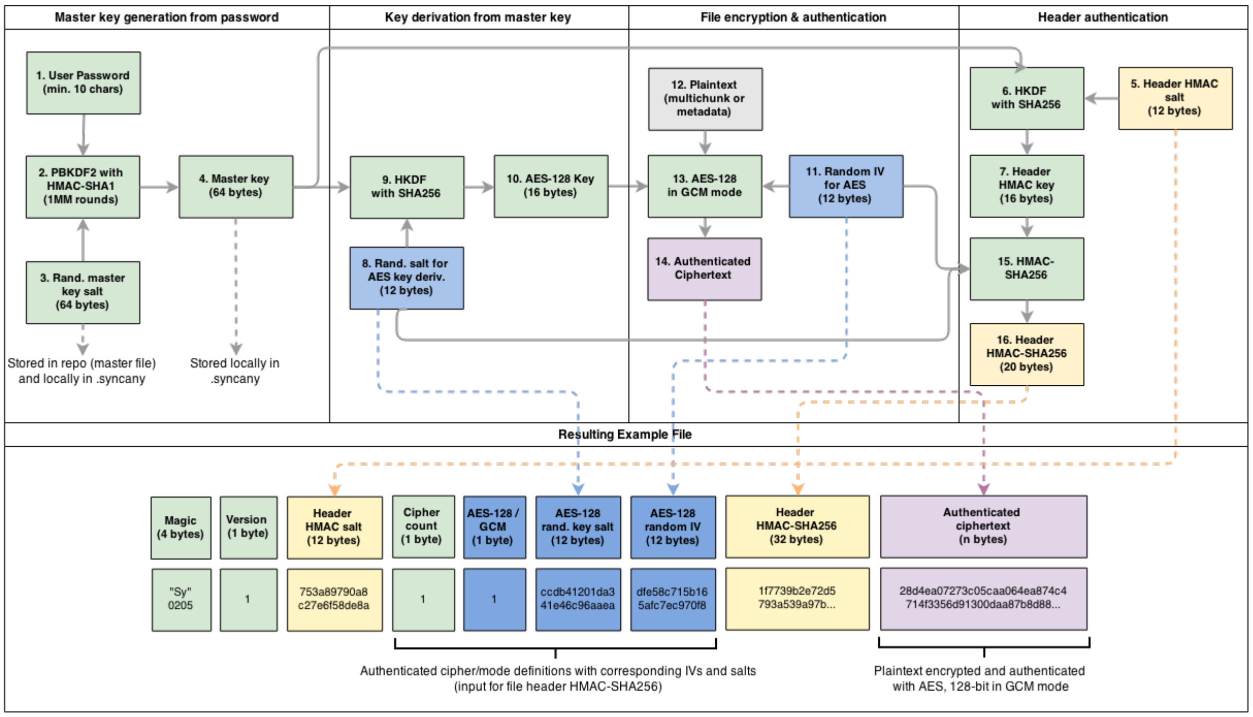
#### Repository Initialization

When a user creates a new repository using `sy init`, Syncany asks the user for the repository password. This password is used to derive a 512-bit **master key** using PBKDF2 with 1 million rounds and a 512-bit random **master salt** (1-4). The master key is stored locally in the `config.xml` file (see *Common Settings and Storage Connection (config.xml)*) – since the local machine is trusted, it is not encrypted. The master salt is stored in plaintext locally (although it is not needed) and on the offsite storage in the `master` file. Since the master salt must be public for when other clients connect to the repository, the `master` file is not encrypted.

#### Encrypting new Files

Whenever new files have to be uploaded, Syncany encrypts these files with **128-bit AES/GCM** using the crypto scheme and file format described below (8-16). File encryption happens with every file that is uploaded (except the `master` file): For instance, during the file index process in `sy up`, Syncany packages new/unknown chunks into multichunks (= ZIP files containing many chunks) and creates a new database file (= metadata). These files are encrypted using the the following scheme:

In the default configuration, Syncany uses one cipher. However, Syncany also supports nesting of ciphers. In this setup, the plaintext is encrypted with the first cipher, and every additional cipher is applied to the ciphertext of the previous cipher. This setup has relatively little research behind it, but we have decided to keep it for the more paranoid users. As of today, Syncany can theoretically nest any authenticated cipher that is supported by the Bouncy Castle Cryptography Provider, but only Twofish/GCM and AES/GCM (both 128-bit or 256-bit) are allowed by Syncany. If you are considering a nested cipher setup, we recommend first switching to **256-bit AES/GCM**.

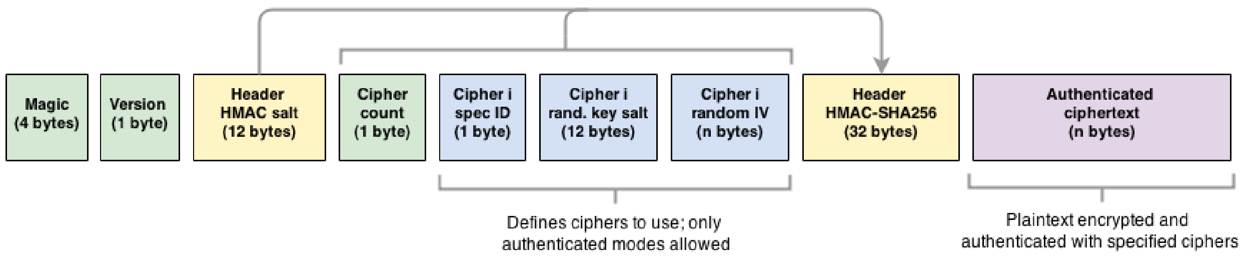


Master key generation from password
Key derivation from master key
File encryption & authentication
Header authentication
1. User Password (min. 10 chars)
2. PBKDF2 with HMAC-SHA1 (1MM rounds)
3. Rand. master key salt (64 bytes)
4. Master key (64 bytes)
Stored in repo (master file) and locally in .syncany
Stored locally in .syncany
9. HKDF with SHA256
10. AES-128 Key (16 bytes)
8. Rand. salt for AES key deriv. (12 bytes)
12. Plaintext (multichunk or metadata)
13. AES-128 in GCM mode
11. Random IV for AES (12 bytes)
14. Authenticated Ciphertext
6. HKDF with SHA256
5. Header HMAC salt (12 bytes)
7. Header HMAC key (16 bytes)
15. HMAC-SHA256
16. Header HMAC-SHA256 (20 bytes)
Resulting Example File
Magic (4 bytes)
Version (1 byte)
Header HMAC salt (12 bytes)
Cipher count (1 byte)
AES-128 / GCM (1 byte)
AES-128 rand. key salt (12 bytes)
AES-128 random IV (12 bytes)
Header HMAC-SHA256 (32 bytes)
Authenticated ciphertext (n bytes)
"Sy" 0205
1
753a89790a8 c27e6f58de8a
1
1
ccdb41201da3 41e46c96aaea
dfe58c715b16 5afc7ec970f8
1f7739b2e72d5 793a539a97b...
28d4ea07273c05caa064ea874c4 714f3356d91300daa87b8d88...
Authenticated cipher/mode definitions with corresponding IVs and salts (input for file header HMAC-SHA256)
Plaintext encrypted and authenticated with AES, 128-bit in GCM mode

The diagram shows an example for the default configuration: For each file to be encrypted, Syncany uses **HKDF with SHA-256** and a **random 96-bit salt** to derive a **128-bit file key** to use for the AES cipher (8-10). In addition to that, a **128-bit random IV** is created and used as input for the cipher (11). Using the 128-bit key and the 128-bit IV, Syncany then encrypts plaintext (= multichunk file, database file, etc.) using the AES cipher in GCM mode.

To reduce improve encryption/decryption performance, Syncany **re-uses file keys up to 100 times** – meaning that up to 100 multichunks or database files are encrypted with the same key. Given that the maximum file size for multichunks is about 4 MB, max. 400 MB might be encrypted with the same key – although typically it's much less. **IVs are never re-used!**

The salts and IVs as well as the cipher configuration itself (here: AES/GCM) is stored in the file header of the crypto file format. Since this information is required to decrypt the files, they are unencrypted. However, to avoid an attack on the clients through header tampering, the header is authenticated using an **HMAC with SHA-256**, using **a 128-bit header key** derived from the master key and a random **96-bit header salt**.


Magic (4 bytes)
Version (1 byte)
Header HMAC salt (12 bytes)
Cipher count (1 byte)
Cipher i spec ID (1 byte)
Cipher i rand. key salt (12 bytes)
Cipher i random IV (n bytes)
Header HMAC-SHA256 (32 bytes)
Authenticated ciphertext (n bytes)
Defines ciphers to use; only authenticated modes allowed
Plaintext encrypted and authenticated with specified ciphers

The resulting **crypto file format** is structured as follows:

- **Magic identifier:** Used to identify Syncany-encrypted files (static `0x53790205`)
- **Crypto format version:** Used to identify the crypto format version (static `0x01`)
- **Header HMAC salt:** Used to derive the HMAC header key with HKDF (to verify the header)
- **Cipher Count:** Defines the number of nested ciphers (default: 2)
- **Cipher Spec ID** Identifies the algorithm and key size used for the first/second/.. cipher
- **Cipher Salt:** Random salt used to derive the cipher-specific file key
- **Cipher IV:** Random IV used as input for the given cipher (size depends on cipher spec ID)
- **Header HMAC:** HMAC calculated over the cipher count and cipher specs.

### Connecting new Clients

When a user connects to an existing repository using `sy connect`, Syncany first downloads the `master` file. This master file contains the unencrypted **master salt** which (in combination with the **repository password**) can be used to derive the **master key**. Using this master key and the salts and IVs contained in the encrypted database and multichunk files, Syncany can create the file keys and thereby decrypt any file.

### `syncany://` Links

After the actual initialization, the `sy init` command creates a so-called `syncany://`-link which can be used by other users to connect to a repository. This link contains the plugin credentials needed to access the repository (e.g. FTP host/user/pass). The link is encrypted using the same crypto format as described above, except that the master salt is included and the link is base58 encoded.

Syncany supports two types of links:

1. **Encrypted links (normal)**: Links prefixed `syncany://storage/1/` are encrypted and can be safely shared via unsecure channels.
2. **Plaintext links (not recommended!):** Links prefixed `syncany://storage/1/not-encrypted/` are not encrypted and should **never be shared via unsecure channels**.

Encrypted links are structured like this: `syncany://storage/1/<master-salt>/<encrypted-config>`. Both `<master-salt>` and `<encrypted-config>` are base58 encoded. The master salt is stored in plaintext and unauthenticated. The encrypted config is stored in the same file format as described above, i.e. using a nested cipher combination of AES and Twofish. When a client attempts to connect to a repository using `sy connect syncany://storage/1/...`, Syncany decrypts uses the master salt and the prompted password to derive a master key, and then uses the master key and the IVs and salts in the encrypted config to derive the actual cipher keys. These keys can then be used to decrypt the storage/connection config.

Plaintext links naturally do not contain a master salt. They are structured like this: `syncany://storage/1/not-encrypted/<plaintext-config>`. The `<plaintext-config>` is simply a base58-encoded representation of the storage/connection config.

**Warning:** Never share an unencrypted/plaintext link over unsecure channels, such as instant messengers or e-mail! If the link contains `not-encrypted`, it is trivial to retrieve the storage credentials from it.

### Crypto Algorithms and Parameters

This chapter sumarizes the algorithms and parameters used by Syncany. This is probably a bit repetetive, but maybe useful for people who don't want to read the entire text:

- Users of a shared folder/repository share a repository password
- Random values are generated using Java's default `SecureRandom` implementation (`/dev/urandom` on Linux, CryptGenRandom on Windows)
- The repository password is used to derive one symmetric key per cipher using PBKDF2 (12 byte salt, 1 million rounds)
- The derived symmetric key(s) are used to encrypt files; each key is reused in max. 100 files (~ 400 MB)
- Cipher algorithms are configurable, but not every cipher is allowed: only AES and Twofish (128/256 bit), only authenticated modes (as of now only GCM; no ECB, CBC, etc.)
- Ciphers are initialized with a random initialization vector (IV), IVs are never reused
- Multiple cipher algorithms can be nested/chained (1-n ciphers), e.g. AES-128 and Twofish-256
- Cipher configurations, IVs and salts are authenticated with an HMAC-SHA256

## HTTPS-only Daemon

The Syncany daemon provides an API and a web interface that can be access over HTTPS (not HTTP!). The API is also available via secure WebSockets.

### Keys and Certificates

The keypair and certificate used for the HTTPS server is generated by Syncany upon the first startup of the daemon. Syncany generates a **2048-bit RSA keypair** and then uses this keypair to **generate a self-signed X.509v3 certificate** with a validity of 5 years. The certificates common name is set to the local hostname, and the organization and org-unit to 'Syncany'. The certificate's **SHA-256 hash** is signed using the RSA private key (signature algorithm):

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1409206372293 (0x1481b3ec7c5)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=localhost, O=Syncany, OU=Syncany
        Validity
            Not Before: Aug 27 06:12:52 2014 GMT
            Not After : Aug 27 06:12:52 2019 GMT
        Subject: CN=localhost, O=Syncany, OU=Syncany
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
```

```
                Public-Key: (2048 bit)
                Modulus:
                    00:a0:43:ca:d6:e6:e9:70:2d:ca:d5:77:ad:e9:3a:
                    1a:50:fe: ...
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         74:7b:a9:22:e3:fb:21:cf:15:3c:ba:11:46:c4:7a:6c:8e:2c:
         f4:aa:cc:27:98:e7: ...
```

The private key and the certificate are stored in a key/trust store. Using your own keypair and certificate is also possible. See *Keys and Certificates* for details.

### Authentication and Authorization

The user authentication and authorization capabilities of Syncany to the web server and REST/WS API are very limited. Syncany provides a simple **HTTP Basic-based user authentication** (but only over HTTPS!). All authenticated users have complete access to the REST/WS API. The user configuration is done via the `daemon.xml` file. See *User Authentication (<users>)*.

## Attack Scenarios

Syncany tries to prevent against a certain threat scenarios. This chapter briefly shows how an adversary might try to attack Syncany. In general, we differentiate between **attacks on the data in transit** and **attacks on the storage provider's side**. Since the local machine is assumed to be secure and data we're trying to protect is not encrypted on the local machine, attacks on the local machine are disregarded.

### Attacking Data Transmissions

An adversary with access to the network infrastructure, e.g. through network monitoring or a man-in-the-middle attack, can either passively monitor the network traffic or actively modify the data being transmitted.

Since Syncany can be used with a many different plugins, **the overall security of the solution strongly depends on the storage plugin**. If, for instance, the FTP plugin is used, the transmitted data can be observed (or even modified) by the adversary, because the FTP protocol does not provide communication security. Similarly, if the WebDAV plugin is used with a HTTP target, the same attack scenario is possible.

However, because Syncany encrypts files before upload, the data being transmitted is of little to no value to the adversary. Even if the data is changed by the adversary, Syncany will detect these changes, because only authenticated ciphers are used – meaning that data confidentiality and integrity is still ensured. When using a plugin without communication security and the adversary can modify the network, **data availibility might be compromised**. An attacker might simply read the storage access credentials and delete the entire repository.

If, however, a plugin is used that provides communicatiobn security, an attacker cannot modify network transmissions and **data availibility is ensured**. Examples for such plugins include the WebDAV plugin with a HTTPS target, the Amazon S3 plugin, or the SFTP plugin. In summation:

- Syncany provides data confidentiality, integrity and availability against attacks on the network if the used plugin provides communication security
- Syncany provides data confidentiality, integrity, **but not availability** against attacks on the network if the used plugin **does not provide communication security**

### Provider-originated Attacks

Syncany stores its data at a central storage. By definition, the provider of that storage has complete access to the data that resides on that storage. If an evil provider takes interest in that data, it is very easy to gain access to it. If, for instance, the owner on an FTP server decides to modify or delete your repository, it is very easy for them to do so. In fact, **Syncany can never provide any protection against a provider-originated attack on data availibility**.

However, similarly to the above mentioned no-communication-security scenario, Syncany still provides data confidentiality and integrity, because files are encrypted in an authenticated mode before upload. A provider might be able to retrieve the encrypted files (or even delete them), but it won't be able to decrypt them.

### Theft of Credentials

One of Syncany's assumptions is that users sharing a repository must trust each other completely. The reason for that is that to access a repository, credentials to the storage are shared. If one of the trusted users were to be tricked into giving up the password, or her laptop were to be compromised, the repository password, the master key or the offsite storage credentials might be in the hand of an attacker.

If the repository password or the master key is retrieved, data confidentiality is completely breached – without other users having a chance of detecting it and without a chance of changing the password. **Syncany can not prevent or detect if the master key or password has been stolen or was used by an adversary.**

If only the storage credentials are retrieved by an adversary, only the availability of data is at risk (same scenario as above).

## Syncany Server Communication

Syncany usually only communicates with the backend storage it is being used with. However, there are very few instances in which it actually calls out to the Syncany server[1]:

- **Listing available plugins**: To list available remote plugins, Syncany queries the Syncany API (at api.syncany.org). Listing plugins is manually initiated via the command line (`sy plugin list`) or when opening the Preferences dialog in the GUI.
- **Downloading/Installing plugins**: When a plugin is installed via `sy plugin install <plugin-id>` (or via the GUI), Syncany retrieves the download location via the Syncany API, and then downloads this plugin from the that location (at get.syncany.org).
- **Checking for application/updates**: To check for new versions of the application and/or plugins, Syncany will query the Syncany API. This can be initiated via `sy update check` or via the GUI once a day.
- **Pub/sub server**: To quickly notify other clients that new data has been uploaded, Syncany subscribes each user to a small pub/sub server at notify.syncany.org:8080 based on Fanout. The data exchanged via this pub/sub server only contains the random repository identifer and is only used to trigger the other clients sync process.
- **Short links**: To shorten the `syncany://` links (using the `sy (init|genlink) --short` option), Syncany will send the long link to the Syncany API and request a short link from there. The link will only be transferred with the `--short` option enabled. Please note that this will link (while heavily encrypted) contains your **encrypted login credentials** (see *syncany:// Links*).

All calls to the Syncany API can be manually overridden by specifying an alternative API endpoint (`--api-endpoint=...`). The pub/sub server can be overridden by the `--announce` command line options or the corresponding configuration setting.

[1] The Syncany server is a rented server managed by the project lead of Syncany. It is located in Germany and hosted by Host Europe GmbH.

## Source Code

All the cryptography related code is implemented in the `org.syncany.crypto` package. Feel free to inspect the code and create a new issue if something doesn't feel right.

## Known Issues and Limitations

- In multiple peer reviews, it has been suggested to drop the cipher nesting in favor of a single cipher. While there is no evidence that a nested cipher is or might be weaker than a single cipher, there is very little literature about it – so it is probably not a good idea. See issue 59.
- As of today, neither the master key nor the password can be changed. See issue 150.

CHAPTER 8

# Configuration

Syncany differentiates between two types of configuration:

- **Folder-specific Configuration**: Each folder that is synchronized and managed by Syncany contains a `.syncany` folder. The files in this folder define the per-folder config options such as the backend storage credentials or the machine name.
- **User-specific Configuration**: The user config defines central settings valid only for the logged in user – this includes proxy configuration or the standby settings, but also daemon-specific configuration such as which Syncany folders are managed by the background daemon, and where the web frontend and REST/WS API is running.

Syncany stores the folder-specific configuration options in `.syncany` of the synchronized folder, and other general user-specific configurations (such as the user and daemon config) in `%AppData%\Syncany\` or `~/.config/syncany/`.

**Contents**

## Folder-specific Configuration

Whenever a new Syncany folder is initialized via `sy init` or `sy connect`, Syncany creates a `.syncany` folder in that directory. This folder marks the folder as a Syncany folder and contains configuration and metadata information. Typically, **you don't have to touch any of these files**, but it doesn't hurt to know a little bit about them:

- `.syncany/config.xml` (main config): Stores client information and the connection details.
- `.syncany/syncany` (repo file): Stores common repository information (repo ID, chunking options).
- `.syncany/master` (master salt file): Stores the salt of the master key (if repo encrypted).
- `.syncany/cache/` (local cache): Cache folder to store temporarily files.
- `.syncany/db/` (local database): Files stored by the local embedded HSQLDB database.
- `.syncany/logs/` (log files): Stores log files created by Syncany. Log files rotate automatically.
- `.syignore` (exclude/ignore files): Lists all the files to be ignored by Syncany.

Since only the `config.xml` and `.syignore` files should be changed manually, the following chapters only describe these files.

### Common Settings and Storage Connection (`config.xml`)

The `config.xml` file is initially created by the `sy init` or `sy connect` command. Changing its settings is typically not necessary – unless your connection/storage details (such as credentials or hostname) change.

#### Options for `config.xml`

| Option | Mand. | Def. | Description |
|---|---|---|---|
| `<machineName>` | yes | *none* | Random local machine identifier |
| `<displayName>` | yes | *none* | Human readable user name |
| `<masterKey>` | no | *none* | Master key used to derive enc. keys |
| `<cacheKeepBytes>` | no | 524288000 | Size of cache in bytes |
| `<connection>` | yes | *none* | Key/value based storage settings |

The `<machineName>` property represents a local machine identifier to technically identify a computer/user. It is purely random and carries no logic. This identifier is created during folder initialization and **should not be changed**.

The `<displayName>` is the human readable user name of the user. It is currently only used locally to create conflict files. As of today, this property is not transferred to other users, but it might be in the future. The property can safely be changed.

The `<masterKey>` represents the master key generated from the repository password. It is required if the repository is encrypted, and it must not be changed. To learn more about the master key, check out the *Security* chapter.

The `<cacheKeepBytes>` property depicts the maximum size of the local `.syncany/cache/` folder (in bytes, default is 500 MB). After each synchronization, the cache is cleaned if it exceeds the keep size. The cache mechanism is least recently used (LRU).

The `<connection>` property represents the *storage plugin* configuration. The `type="<plugin-id>"` attribute defines the plugin used to synchronize this folder. The definition of these properties is described in the corresponding plugin sub-chapter.

### Example `config.xml`

```
<config xmlns="http://syncany.org/config/1">
        <machineName>PCiqLdSQiampovfBfSZZ</machineName>
        <displayName>pheckel</displayName>
        <masterKey salt="51d32e99230581e2..." key="ba2f6725d2ce7d90822..."/>
        <connection type="ftp">
                <hostname>ftp.example.com</hostname>
                <username>armin</username>
                <password encrypted="true">87abc68afe1428319fad...</password>
                <path>/syncany/repo2</path>
                <port>21</port>
        </connection>
        <cacheKeepBytes>524288000</cacheKeepBytes>
</config>
```

## Excluding/Ignoring Files and Folders (`.syignore`)

The `.syignore` file allows you to ignore certain files and folders from the synchronization process. It must be created manually by the user if any exclude/ignore logic is desired. The file resides in the root of the managed folder and is itself synchronized to other clients using Syncany.

**Note:** As of today, new entries in the `.syignore` file are not picked up by Syncany (files are not ignored!) if the to-be-ignored-file has already been synchronized. We are aware that this is not a desired behavior and are working on it to fix it.

The file has a simple line-based structure, in which each line represents a path to be ignored by Syncany. The file supports the typical wildcards (`*` and `?`) as well as regular expression based patterns:

- **Wildcard-based exclusions:** `*` matches any amount of characters (including none), `?` matches exactly one character.
- **Regular expression based exclusions:** Lines prefixed `regex:` exclude files matching the given regular expression.

If `C:\Users\Steffen\Syncany` is the managed Syncany folder, the following file (if located at `C:\Users\Steffen\Syncany\.syignore`) will ignore files/folders ending with `.bak`, file/folders named `.git` as well as files/folders matching the regular expression `private/20[0-9]{2}` (e.g. `private/2099` or `private/2000`):

```
*.bak
.git
regex:private/20[0-9]{2}
```

## User-specific Configuration

The user config defines central settings valid only for the logged-in user. Unlike the folder-specific settings, the user configuration settings apply to the entire user. There are three general categories of user-specific configuration files:

- **General User Configuration** (`userconfig.xml`): Define central user-specific config options such as proxy settings, standby settings or other system properties.
- **Daemon Configuration** (`daemon.xml`): Define settings specific to the Syncany background process (the daemon), such as which folders are managed by the daemon.
- **GUI Configuration** (`gui.xml`): Define settings specific to the Syncany graphical user interface (only present if GUI is installed).

The configuration can be found at `%AppData%\Syncany\` (Windows) and at `~/.config/syncany/` (Linux).

## User Config (`userconfig.xml`)

The `userconfig.xml` config file is a defines global user config settings – valid only for this user, but regardless of whether or not Syncany is run in daemon mode or manually. The options are pretty limited are right now. More config options will probably be added in future releases.

### Options for `userconfig.xml`

| Option | Mand. | Def. | Description |
|---|---|---|---|
| `<configEncryptionKey>` | yes | *none* | Encryption key to encrypt repo access credentials |
| `<systemProperties>` | no | *none* | Set any Java system properties (e.g proxy) |
| `<preventStandby>` | no | false | Prevent standby/shutdown during sync |
| `<maxMemory>` | no | 512M | Limit memory usage of Syncany |

The `<configEncryptionKey>` option is used to encrypt sensitive values in the *config.xml* file, e.g. the (S)FTP/WebDAV password, the Amazon S3 / Dropbox access token, and so on.

The `<systemProperties>` option allows you to set Java system properties via the Syncany configuration. Any of the `<property>` options will be passed to Java's `System.setProperty()` method. This can be used to set proxy settings, log settings, and so on.

If the `<preventStandby>` option is set to `true`, Syncany will make sure that your system doesn't go into standby/hibernate if the synchronization process is run. This option will not prevent your system from going to sleep if no upload/download process is taking place. Since this option might also prevent the screensaver or screen lock, it is not enabled by default.

The `<maxMemory>` option can be used to limit Syncany's memory usage to the given value. Example values are `1G` (1 GB) or `500M` (500 MB). By default, Syncany will limit the max. memory value to `512M` (512 MB). The value will be passed on to the Java Virtual Machine (`-Xmx`).

### Useful System Properties

This is a non-exhaustive list of useful system properties that can be used in the above mentioned `<systemProperties>` option. To add an option, simply add a property tag:

```xml
<property name="property-name">property value</property>
```

| System Property | Description |
|---|---|
| **http.proxyHost** | Sets HTTP proxy hostname |
| **http.proxyPort** | Sets HTTP proxy port |
| **http.proxyUser** | Sets HTTP proxy username (if proxy needs authentication) |
| **http.proxyPass** | Sets HTTP proxy password (if proxy needs authentication) |
| **https.proxyHost** | Sets HTTPS proxy hostname |
| **https.proxyPort** | Sets HTTPS proxy port |
| **https.proxyUser** | Sets HTTPS proxy username (if proxy needs authentication) |
| **https.proxyPass** | Sets HTTPS proxy password (if proxy needs authentication) |
| **socksProxyHost** | Sets SOCKS proxy hostname |
| **socksProxyPort** | Sets SOCKS proxy port |
| **java.net.socks.username** | Sets SOCKS proxy username (if proxy needs authentication) |
| **java.net.socks.password** | Sets SOCKS proxy password (if proxy needs authentication) |
| **org.syncany.test.tmpdir** | Developer property: Uses the given folder for Syncany unit tests |

### Example `userconfig.xml`

This example shows how to set the HTTP and HTTPS proxy for all HTTP/HTTPS-traffic by Syncany. In particular, this includes traffic to the Syncany Plugin API and communication by the *WebDAV plugin*. The example furthermore shows how to enable the standby/hibernate prevention.

```
<userConfig xmlns="http://syncany.org/userconfig/1">
   <preventStandby>true</preventStandby>
   <systemProperties>
      <property name="http.proxyHost">your.proxy.host.tld</property>
      <property name="http.proxyPort">8080</property>
      <property name="https.proxyHost">your.proxy.host.tld</property>
      <property name="https.proxyPort">8080</property>
   </systemProperties>
   <configEncryptionKey salt="87d32e99230581e2..." key="652f6725d2ce7d90822..."/>
</userConfig>
```

## Daemon Config (`daemon.xml`)

The main purpose of the daemon configuration is to tell the Syncany daemon (started by `sy daemon start`) what folders should be monitored and automatically synced whenever something changes.

### Options for `daemon.xml`

| Option | Mand. | Def. | Description |
|---|---|---|---|
| `<folders>` | yes | *none* | Folders managed by the daemon |
| `<webServer>` | yes | *none* | Internal web server parameters |
| `<users>` | yes | *none* | Log-in users for web server and API |

The `<folders>` option can contain multiple `<folder>` definitions, each of which represent a Syncany folder managed by the daemon. To add a new Syncany folder, simply initialize or connect to a repository (using `sy init` or `sy connect`) and add the folder here. Then restart the daemon. Find details to this option below in *Managed Folder Configuration (<folders>)*.

The `<webServer>` option controls the internal Syncany web server (bind port and address, certificates). The web server is used for the web interface as well as for the Syncany API. Find details to this option below in *Web Server Configuration (<webServer>)*.

The `<users>` option defines the users that can access the web interface and the API. Each `<user>` has full read/write access to the API and all managed folders. Find details to this option below in *User Authentication (<users>)*.

### Managed Folder Configuration (`<folders>`)

**Note:** We're currently still in an alpha version of Syncany and the options inside the `<folder>` tag change more often than we desire. Please forgive us for not documenting all of the options. You might want to check out the `daemon-example.xml` file to see all available options.

The `<folders>` tag can contain multiple `<folder>` tags, each of which has a vast amount of configuration options. Typically you don't need to touch any of them. To see a full example (including all available options), see *Example 2: Complex daemon.xml*.

The `<path>` option defines the local path to the Syncany folder, and the `<enabled>` tag lets you enable/disable folders.

The `<watch>` tag defines the behavior for the internal watch operation. That includes the behavior of the index/upload operation (`<up>`), the download/apply operation (`<down>`) as well as the periodic cleanup mechanism (`<clean>`). It also includes general settings about the local file system watcher (`<watcher>`) and the central Syncany pub/sub server (`<announcements>`).

### Web Server Configuration (`<webServer>`)

The internal web server is used to serve the Syncany REST and WebSocket API, as well as the web interface (if a *web interface plugin* is installed). Both API and web interface are HTTPS-only, meaning that no HTTP traffic is accepted and that all communication is encrypted.

The certificate used by the internal web server is automatically generated by default. Syncany creates a self-signed X.509 certificates based on a generated RSA 2048-bit keypair. The common name (CN) in the certificate is the local hostname by default, but can be changed if needed.

**Note:** As of today, providing your own certificates and keypair is possible, but not as easy as it should be: First disable the `<certificateAutoGenerate>` option, and then import your keypair into `keystore.jks` and your certificate into `truststore.jks`. Also see *Keys and Certificates*.

| Option | Mand. | Def. | Description |
|---|---|---|---|
| `<enabled>` | yes | true | Defines whether the web server will start |
| `<bindAddress>` | yes | 127.0.0.1 | Address to which the server socket is bound |
| `<bindPort>` | yes | 8443 | Port to which the server socket is bound |
| `<certificateAutoGenerate>` | yes | true | Regenerate certificate if common name changed. |
| `<certificateCommonName>` | yes | *hostname* | Common name in the server certificate |

The `<enabled>` option can switch off the web server entirely, if the option is set to `false`. However, if the webserver is disabled, neither REST/WS API nor the web interface are available.

The `<bindAddress>` and `<bindPort>` options define to which IP address, i.e. network interface, and port the server will be bound. For security reasons , the web interface and API is only bound to a local address (`127.0.0.1`) and is therefore not reachable externally. To make the API and web interface publicly available, set the bind address to `0.0.0.0` (or a specific IP address).

The `<certificateAutoGenerate>` option automatically (re-)generates a keypair and a self-signed certificate for the Syncany web server using the common name (CN) defined in `<certificateCommonName>`. Whenever

<certificateCommonName> is changed (and <certificateAutoGenerate> is set to true), Syncany will re-generate a new keypair/certificate.

### User Authentication (<users>)

The user authentication capabilities of Syncany to the web server and REST/WS API are very limited. Syncany provides a simple "HTTP Basic"-based user authentication – using users defined in the daemon.xml config file.

All users provided in the <users> option have full read/write access to the Syncany web interface and REST/WS API. As of today, there are no authorization mechanisms at all:

```
<users>
   <user>
      <username>Pim</username>
      <password>IOgotcpZzNPh</password>
   </user>
   <user>
      <username>Philipp</username>
      <password>plaintextpassword</password>
   </user>
</users>
```

In the example, users Pim and Philipp have the same access rights. Both can access the web interface and execute any REST/WS request.

### Example 1: Simple daemon.xml

```
<daemon xmlns="http://syncany.org/daemon/1">
   <webServer>
      <enabled>true</enabled>
      <bindAddress>0.0.0.0</bindAddress>
      <bindPort>8443</bindPort>
      <certificateAutoGenerate>true</certificateAutoGenerate>
      <certificateCommonName>platop</certificateCommonName>
   </webServer>
   <folders>
      <folder>
         <path>/home/pheckel/Syncany</path>
         <enabled>true</enabled>
      </folder>
   </folders>
   <users>
      <user>
         <username>admin</username>
         <password>IOgotcpZzNPh</password>
      </user>
   </users>
</daemon>
```

### Example 2: Complex daemon.xml

```
<daemon xmlns="http://syncany.org/daemon/1">
   <webServer>
      <enabled>true</enabled>
      <bindAddress>0.0.0.0</bindAddress>
      <bindPort>8443</bindPort>
      <certificateAutoGenerate>true</certificateAutoGenerate>
      <certificateCommonName>platop</certificateCommonName>
   </webServer>
   <folders>
      <folder>
         <path>/tmp/repo4</path>
         <enabled>true</enabled>
         <watch>
            <interval>120000</interval>
            <announcements>true</announcements>
            <announcementsHost>notify.syncany.org</announcementsHost>
            <announcementsPort>8080</announcementsPort>
            <settleDelay>3000</settleDelay>
            <cleanupInterval>3600000</cleanupInterval>
            <watcher>true</watcher>
            <up>
               <status>
                  <forceChecksum>false</forceChecksum>
               </status>
               <forceUploadEnabled>false</forceUploadEnabled>
            </up>
            <down>
               <conflictStrategy>RENAME</conflictStrategy>
               <applyChanges>true</applyChanges>
            </down>
            <clean>
               <status>
                  <forceChecksum>false</forceChecksum>
               </status>
               <force>false</force>
               <removeOldVersions>true</removeOldVersions>
               <maxDatabaseFiles>15</maxDatabaseFiles>
               <minSecondsBetweenCleanups>10800</minSecondsBetweenCleanups>
            </clean>
         </watch>
      </folder>
   </folders>
   <users>
      <user>
         <username>admin</username>
         <password>IOgotcpZzNPh</password>
      </user>
   </users>
</daemon>
```

## GUI Config (`gui.xml`)

The GUI config file `gui.xml` is only present if the graphical user interface is installed and has been started at least once. Its purpose is to store GUI-specific options only. All of the options are **meant to be set by the GUI**.

### Options for `gui.xml`

| Option | Mand. | Def. | Description |
|---|---|---|---|
| `<tray>` | no | `DEFAULT` | Tray icon engine (Linux only) |
| `<theme>` | no | `DEFAULT` | Icon set to use for the tray |
| `<startup>` | no | `false` | Run Syncany GUI at startup / user login |
| `<notifications>` | no | `true` | Show (or don't show) notifications |

The `<tray>` option defines the tray backend used to display Syncany's tray icon. Possible values are `DEFAULT`, `APPINDICATOR` and `OSX_NOTIFICATION_CENTER`. If `DEFAULT` or no value is set, Syncany will try to automatically determine the backend to use for displaying the tray icon. On Linux, the Syncany GUI will use the Python-based app indicator script if the Unity desktop is detected. On Mac OSX, it will try to use the OSX notification center.

The `<theme>` option controls the icon theme used by the tray icon. Possible values are `DEFAULT` and `MONOCHROME`. If `DEFAULT` is given, the default color theme will be used for all operating systems except for Mac OSX. On Mac OSX, the `MONOCHROME` theme is used.

The `<startup>` option indicates whether the Syncany GUI is started when the user logs in, i.e. on startup. Note that changing this value in an editor does not change the autostart settings of your system. Only set this value via the graphical user interface.

The `<notifications>` option defines if sync notifications are displayed to the user. Notifications are typically displayed if new files have been added or changed.

### Example: GUI config `gui.xml`

```
<gui>
   <tray>DEFAULT</tray>
   <theme>DEFAULT</theme>
   <startup>true</startup>
   <notifications>true</notifications>
</gui>
```

## Keys and Certificates

Syncany maintains its own user-specific key store for private keys and trust store foreign X.509 certificates. Both key store and trust store are used by the internal web server as well as by plugins that use SSL/TLS (e.g. the *WebDAV plugin*).

Both files are located at the user-specific configuration directory at `~/.config/syncany/` (Linux) or `%AppData\Syncany\` (Windows). The files are stored in the **Java Key Store** (JKS) format. No password is used to protect the key/trust store. To analyze this file and its entries, you may use the `keytool` util.

### Private Keys (`keystore.jks`)

As of today, Syncany only stores one entry in the `keystore.jks` file – namely the private key part of the RSA keypair used to serve the HTTPS API and web interface. As described in *Web Server Configuration (<webServer>)*, the keypair is generated by Syncany whenever the common name in the daemon config is changed (unless the `<certificateAutoGenerate>` option is disabled).

In normal situations, you should not have to alter the key store file at all. To use your own keypair and certificate, please see *Using your own Keypair and Certificate*.

## Trusted Certificates (`truststore.jks`)

Syncany's user-specific trust store holds trusted X.509 certificates of remote servers. This trust store is mainly used by plugins that communicate via SSL/TLS (such as the *WebDAV plugin*). Whenever `sy connect` or `sy init` is called with a plugin that uses TLS/SSL, Syncany will ask the user to confirm certificates that are not in the trust store. After the user confirms, these certificates are added to the trust store, so that the next time the server is contacted, no user query is necessary.

```
Unknown SSL/TLS certificate
---------------------------
Owner: CN=*.syncany.org, OU=Domain Control Validated
Issuer: CN=GlobalSign Domain Validation CA - SHA256 - G2, O=GlobalSign nv-sa, C=BE
Serial number: 1492271418628120790652059091142976109636803
Valid from Mon Apr 14 23:01:38 CEST 2014 until: Wed Apr 15 23:01:38 CEST 2015
Certificate fingerprints:
 MD5:  60:FB:F7:F1:E1:9E:D6:74:06:41:03:01:16:D6:19:D3
 SHA1: DC:A8:5F:FA:1D:9D:92:A7:1C:8E:22:C6:43:9B:96:9E:62:13:C7:25
 SHA256: 84:DF:92:99:86:15:AF:A6:8D:EC:74:5C:13:BE:18:75:BC:08:34:...

Do you want to trust this certificate? (y/n)?
```

In normal situations, you should not have to alter the trust store file at all. You may, however, need to do that if you want to your own keypair and certificate. See *Using your own Keypair and Certificate* for details.

## Using your own Keypair and Certificate

**Note:** Using your own keypair and certificate is not as easy as it should be. Please let us know if you have a better way to do that.

To use your own keypair and certificate for Syncany's internal web server, follow the following steps. You can use a tool like `keytool` to do that:

1. Set the `<certificateAutoGenerate>` option in the `daemon.xml` to `false` (see *Web Server Configuration (<webServer>)*).
2. Replace the only private key entry (and the associated certificate) in the `keystore.jks` file with your own private key and certificate. The private key entry must be the only entry in the file.
3. Import your own certificate in the `truststore.jks`. You don't have to remove any entries from this file.

After restarting the daemon, Syncany should use your own certificate for the internal web server. You can verify that by opening the browser and checking the certificate.

CHAPTER 9

# FAQ

We generally talk a lot about Syncany, and whenever we explain the idea, people tend to ask the same questions. Here are a few of them – hopefully with satisfying answers.

**Contents**

## General questions

### What is Syncany?

Syncany is open source Dropbox-like file sync tool. It syncs files between computers either manually or automatically. Users can define certain folders of their computers and keep this folder in sync with friends.

So what make Syncany **awesome**?

- Use-your-own storage (FTP, S3, WebDAV, NFS, Samba/Windows file share, ...)
- Files are encrypted before upload (don’t trust anybody but yourself)
- Files are deduplicated locally (massive space savings on remote storage)

What Syncany is **not**:

- A service. You don’t pay for it. You don’t get storage from us.
- Usable for critical files. It’s alpha software. Don’t do it!
- A large network of interconnected machines. You can’t share folders with strangers. You must trust the persons you share files with.
- A fully fledged version control system.

### Why make something new? How is it different from X?

If there were something like Syncany, we wouldn’t be developing it. There are a few alternatives, but they all have a different focus, are not open source, have no versioning, cannot use arbitrary storage or don’t encrypt locally. As far as we know, no tool has this exact feature set.

Let’s pick the normal contenders: Dropbox, Wuala, SparkleShare, BitTorrent Sync, ownCloud, rsync, SyncThing.

And here’s how to filter them out: Dropbox, Wuala, and BitTorrent Sync are not open source, ownCloud, rsync and Ubuntu One don’t support encryption. SyncThing is bound to the peer2peer concept.

### Is there an App/Web-App/GUI for it?

Yes! You can install it using `sy plugin install gui`.

There is an outdated (and currently broken) proof of concept web interface.

### When will it be available? When is the release date?

We don't promise anything, because this is a spare time project. We are slowly moving towards beta and after that it will be a lot of testing before we can remove the beta label.

### How can I help?

If you're a developer, try out the code, review it and contact us on GitHub if you want to help. Check out the issues to find out more. Check out the wiki page for more ways to help.

### Does it support multiple users?

You can connect to a single repo from many different machines, but there is no role concept or access control for users. All users have the same rights, meaning there is no administrator and once a user is connected he or she can easily delete all repo data if they want to.

### Does it work on Windows/Linux/Mac?

The majority of Syncany developers work on Linux, but we are also testing it on Windows. We've done short cross-platform tests and it seems to work okay. We'll test that even more in the future.

### What shouldn't I use Syncany for?

Here are some things Syncany isn't good at, and things you shouldn't use Syncany for:

- **Critical files**: At the moment, **do not use Syncany for critical files**. The software is still alpha software!
- **Low CPU environments**: Syncany uses up quite a bit of CPU power when it is indexing files. So in general, any situation where the client has little computation power.

### Can I use Syncany to sync two unencrypted folders? Can I mirror two folders?

No this is not possible. Syncany stores data in a packed repository form. While this might seem as a disadvantage at first, it makes great things possible:

- Save space: Using the deduplication mechanisms in Syncany, you can save a lot of remote storage space (>90% space savings)
- File versioning: Syncany versions files, so if you overwrite or lose a file, you can restore it very easily
- Huge files: Syncany splits files into chunks, so it doesn't matter how large the original file is.

To sync two folders directly, you can use rsync or Unison.

### Where does the name Syncany come from?

The working title was Clonebox for a while. Then Philipp tried to find a name via Amazon Mechanical Turk. That didn't work. Philipp's girlfriend finally came up with the nice name Syncany. It combines the two works *synchronize* and *anything*.

### How do you pronounce Syncany?

- In phonetic symbols: si■kni
- Like in this audio file.

### SyncAny? Sync Any? Syncanny?

No. Syncany.

## Technial and security questions

### Why is it written in Java?

That's easy: Because we speak Java!

### Can you write a plugin for X?

People often ask about plugins for other storage backend – examples include Hazelcast, OpenStack Swift, Box.net, Amazon Cloud Drive, etc.

We're currently concentrating on the core functionality – so plugins are not very high on the list. However, plugins are very easy to develop; just 200 lines of code or so... :-) If you want, you can of course develop the plugin yourself.

Plugins can be independently developed and deployed in other repos.

### What is the purpose of storing the master salt in the repo? Isn't it already included in the Syncany link?

The master salt is stored in the repo in case we do not use (or cannot use) Syncany links. When we simply do 'sy connect', we don't have the salt anywhere, so it must be retrieved from the server.

### What exactly is stored in the 'syncany' file in the repo? Is it needed?

The Syncany file stores information about the chunking mechanisms used. It is currently only used to check if the password is correct (see if it decrypts and deserializes correctly, but it will be used in the future to store repository-specific information.

### What file meta data (permissions, access time, etc.) are preserved in the sync?

As of today, for each file version, we store the following metadata:

- Version (1, 2, ...)
- Type (FILE, FOLDER, SYMLINK)
- Status (NEW, CHANGED, ...)
- Relative path (e.g. 'Documents/Hello.txt')
- Symlink target (if type is SYMLINK, e.g. 'Documents/Hello-orig.txt')

- Size (in bytes)
- Last modified date/time (second-accuracy only due to file system restrictions)
- Updated date/time
- DOS permissions (archive, read-only, system, hidden)
- POSIX permissions (rwxrwxrwx)

The following things are *not* stored:

- Hardlinks are NOT stored/detected (not efficient), but data is not stored twice
- POSIX uid/gid are NOT stored
- ACLs are NOT stored
- Extended attributes are NOT stored (see #392)
- Access time and create time are NOT stored

You can see the metadata yourself by running `sy init --no-encryption --no-compression` and then looking at the *database-** files in the repository in a XML or text editor.

### How can I edit the connection details, or reconnect to a repository?

If your backend storage credentials change (e.g. FTP user or password), it might be necessary to reconnect a local folder to a remote repository. As of today, you *cannot* edit a connection via the Syncany commands or GUI. As of now, you can manually change the `.syncany/config.xml` file to achieve that. There already is a ticket to do a "sy reconnect" command (or something similar on GitHub.

## Common errors

### Error message "Could not generate DH 4096/2048 bits keypair"

If you see the error message "Could not generate DH 4096 bits keypair" or "Could not generate DH 2048 bits keypair" (or a similar message) during the `sy (init|connect)` operation, the likely cause is that Java 7 only supports DH keys up to 1024 bits (due to a bug in Java 7 as indicated in this StackOverflow post) and Java 8 only supports DH keys with up to 2048 bits. As noted in GitHub issue #483, upgrading to Java 8 will allow DH keys with up to 2048 bits. Keys with 4096 bits are not yet supported in Java 7 or 8. The only "solution" is to downgrade the server's TLS/SSL security parameters (cipher suites) to accept/downgrade to DH 2048 bit keys.

CHAPTER 10

# Appendix A: Manual Pages

This appendix lists all the manual/help pages available for the `sy` command, its sub-commands. These help pages are available by calling `sy <action> --help` or `man sy-<action>` (Linux only).

**Contents**

## sy / syncany

```
NAME
  sy - secure file sync software for arbitrary storage backends

SYNOPSIS
  sy [-l|--localdir=<path>] [--log=<path>] [-v]
     [-vv] [--loglevel=<level>] [--print]
     [-d|--debug] [-h|--help] <command> [<args>]

DESCRIPTION
  Syncany is an open-source cloud storage and filesharing application. It
  allows users to backup and share certain folders of their workstations
  using any kind of storage.

  Main sync commands:
    init       Initialize the current folder as a Syncany folder
    connect    Connect the current folder to an existing Syncany repository
    up         Detect local changes and upload to repository
    down       Detect remote changes and apply locally

  Other commands:
    status     Detect and print local changes
    ls-remote  Detect and print remote changes
    ls         List and filter the current and past file tree.
    cleanup    Remove old versions from the local database and the repo
    restore    Restore the given file paths from the remote repository
    genlink    Create a syncany:// link from an existing local folder
    plugin     List, install and remove storage backend plugins
    update     Manage and check for application updates
    daemon     Start and stop the background process (daemon)
    watch      Automatically sync the local folder (in the foreground)

  Short command descriptions and options can be found below. Detailed
  explanations can be queried with `sy <command> --help`.

  An extensive user guide with many examples is also available on the
  Syncany website: https://syncany.org/r/userguide

GLOBAL OPTIONS
  -l, --localdir=<path>
      Use <path> instead of the current directory as local sync folder.
      Syncany searches for a '.syncany' folder in the given and all parent
      directories.

  -d, --debug
      Sets the log level to ALL, and print the log to the console.
      Alias to: --loglevel=ALL --print

  -h, --help
      Print this help screen and exit.

  -v, -vv
      Print application version exit.

  --log=<path>
      Log output to the file given by <path>. If - is given, the
      output will be logged to STDOUT (default).
  --loglevel=<level>
      Change log level to <level>. Level can be either of the
      following: OFF, SEVERE, WARNING, INFO, FINE, FINER, FINEST, ALL

  --print
      Print the log to the console (in addition to the log file).

AUTHORS
  Written by Philipp C. Heckel and many others

REPORTING BUGS
  Please report bugs to the GitHub issues page at
     https://www.github.com/syncany/syncany/issues

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy cleanup

```
NAME
  sy-cleanup - remove old versions and free remote disk space

SYNOPSIS
  sy cleanup [-o | --delete-older-than=<relative-time>] [-f | --force]
             [-I | --no-delete-interval] [-O | --no-delete-older-than]
             [-T | --no-temp-removal] [<status-options>]

DESCRIPTION
  This command performs different operations to cleanup the local database as
  well as the remote store. It removes old versions from the local database,
  deletes unused multichunks (if possible) and merges remote database files,
  if necessary.

  Remove old file versions: File versions are deleted by two criteria. The
  first is if it is older than 30 days (configurable with -o, disable with -O).
  The second is an interval based strategy, to keep the version history
  readable. By default, one version is kept per minute in the last hour,
  one verison is kept per hour in the last three days and one version
  per day is kept in the last month. This strategy can be disabled with -I.

  Merge remote databases: The remote databases of the local client are merged
  together if there are more than 15 remote databases per client. The purpose
  of this is to avoid endless amounts of small database files on the remote
  storage and a quicker download process for new clients. In addition,
  databases are merged whenever versions are removed.

  This command uses the 'status' and 'ls-remote' commands and is only executed
  if there are neither local nor remote changes.

OPTIONS
  -o, --delete-older-than=<relative-time>
    Sets the time that cleanup waits before deleting all versions that are
    older than this time. Until this time, deleted files can still be restored.
    Cleanup will fully delete files that were deleted longer ago in the past
    than this amount of time and they will be gone permanently. In addition,
    any versions of files which are not the current version, and older than
    this threshold will also be deleted.
    Default is 30 days (30d).

    Relative time format: <value><unit>, for which <value> may be any
    floating point number and <unit> may be any of the following: s(econds),
    m(inutes), h(ours), d(ays), w(eeks), mo(nths), y(ears). Units may be
    shortened if they are unique. Examples: 5h30m or 1y1mo2d

  -I, --no-delete-interval
    Turns off the version removal strategy based on intervals.
    By default the following policy is applied:
    For the last hour,         the last version from each minute is kept.
    For the last three days,   the last version from each hour   is kept.
    For the last thirty days, the last version from each day    is kept.

    With this option, only old file versions (-o) will be deleted.

  -O, --no-delete-older-than
    Turns off the removal of old versions for the command. If this is set, this
    command will not delete file versions purely because they are old.

  -T, --no-temp-removal
    Turns off the removal of leftover temporary files for the command. If this
    is set, this command will leave temporary files on the offsite storage
    untouched.

  -f, --force
    Forces a the cleanup, even if the time between cleanups (3 hours) has not
    passed. Use this option only if a cleanup is absolutely necessary and you
    know what you are doing.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy connect

```
NAME
  sy-connect - connect to an existing Syncany repository

SYNOPSIS
  sy connect <syncany-link>
             [-n | --add-daemon] [--password]

  sy connect [-P | --plugin=<plugin>] [-o | --plugin-option=<key=value>]
             [-n | --add-daemon] [--password]

DESCRIPTION
  This command connects to an existing remote repository and initializes
  the local directory.

  The command can be called as follows:
```

```
  1. Using a syncany:// link generated by either 'init' or 'genlink',
     the command connects to the repository given in the link. If the link
     is encrypted, the link/repo password must be entered.

  2. If no link is given, the command acts like 'init', i.e. it queries the
     user for storage plugin and connection details of the repository to
     connect to.

  Once the repository is connected, the initialized local folder can be synced
  with the newly created repository. The commands 'up', 'down', 'watch', etc.
  can be used. Other clients can then be connected using the 'connect' command.

OPTIONS
  -P, --plugin=<plugin>
    Selects a plugin to use for the repository. Local files will be synced via
    the storage specified by this plugin.

  -o, --plugin-option=<key=value> (multiple options possible)
    Sets a plugin-specific setting in the form of a key/value pair. Each
    plugin defines different mandatory and optional settings. At least, all
    mandatory settings must be specified by this option. All mandatory and
    optional settings can be listed using the 'plugin' command.

  -n, --add-daemon
    The initialized local folder is automatically added to the daemon
    configuration for automatic synchronization if this option is used.

  --password=<password>
    DO NOT USE THIS OPTION. Set the password to decrypt the repository.
    This option shouldn't be used, because the password might be visible to
    other users or be stored in history files.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy daemon

```
NAME
  sy-daemon - starts and stops the background process (daemon)

SYNOPSIS
  sy daemon (start|stop|reload|restart|status|force-stop)

  sy daemon list

  sy daemon add <folder-path>

  sy daemon remove (<folder-path> | <folder-index>)

DESCRIPTION
  This command manages the Syncany background process (aka the daemon). It can
  start and stop the daemon, display the status and reload the daemon
  configuration.
  With the actions `list`, `add` and `remove`, the command furthermore manages
  the Syncany folders controlled by the daemon. Controlled folders are synced
  automatically when the daemon is running.

  This daemon can be started with `sy daemon start`. Once it is running, all
  registered folders are monitored for changes and remote changes are
  automatically applied to the local folder(s). All of these actions happen
  in the background, without the need for any intervention.

  The daemon is configured using the `daemon.xml` file  at
  ~/.config/syncany/daemon.xml.

OPTIONS
  start
    Starts the background process (if it is not already running).

  stop
    Stops the background process (if it is running).

  reload
    Reloads the daemon configuration without restarting the proces.

  restart
    Stops, then starts the daemon again.

  status
    Displays the status and the process ID (PID) of the daemon.

  force-stop
    Forces the process to stop. Do not use this unless absolutely necessary!

  list
    Lists the folders managed by the daemon.

  add <folder-path>
    Adds the given folder to the daemon configuration. The added folder will
    be managed by the daemon after the config has been reloaded, or the
    daemon is restarted.

  remove (<folder-path> | <folder-index>)
    Removes the given folder from the daemon configuration. The argument can
    either be the full path of the folder or the index of the folder (as
    printed by the `list` action). Changes will be applied after a restart
    of the daemon.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy down

```
NAME
  sy-down - fetch remote changes from Syncany repository and apply locally
```

```
SYNOPSIS
  sy down [-C | --conflict-strategy=<rename|ask>] [-A | --no-apply]

DESCRIPTION
  This command detects changes made by other clients and applies them
  locally. If there are remote changes, the command downloads the relevant
  metadata, evaluates which multichunks are required and then downloads them.
  It then determines what files need to be created, moved, changed or deleted,
  and performs these actions, if possible.

  In some cases, file conflicts may occur if the local file differs from the
  expected file. If that happens, this command can either automatically rename
  conflicting files and append a filename suffix, or it can ask the user what
  to do.

  To determine the remote changes, the 'ls-remote' command is used.

OPTIONS
  -A, --no-apply
    All local file system actions are skipped, i.e. the local folder will not
    be changed. Only the new/unknown database versions will be downloaded and
    persisted to the database.

  -C, --conflict-strategy=<rename|ask>
    Chooses the conflict resolve strategy if a local file does not match the
    expected local file (as per the local database). The conflict strategy
    describes the behavior of this command.

    * The 'rename' strategy automatically renames conflicting files to a
      conflicting file name (e.g. "Italy (Philipp's conflicted copy).txt").

    * The 'ask' strategy lets the user decide whether to keep the local file,
      apply the remote file, or create a conflicting file (as above).

    The default strategy is 'rename'.
    The 'ask' strategy is currently NOT implemented!

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy genlink

```
NAME
  sy-genlink - generate Syncany link for initialized local directory

SYNOPSIS
  sy genlink [-s | --short] [-m | --machine-readable]

DESCRIPTION
  This command creates a Syncany link (syncany://..) from an existing local
  folder. The link can then be sent to someone else to connect to the
  repository.

  Syncany links contain the connection information of the storage backend,
  so in case of an FTP backend, host/user/pass/etc. would be contained in
  a link. If the link is shared, be aware that you are giving this information
  to the other users.

OPTIONS
  -s, --short
    The generated syncany:// link will be shortened using the Syncany link
    shortener service. This option stores the encrypted link on the Syncany
    servers. The option does not work if the repository is not encrypted.

  -m, --machine-readable
    Only prints the link and leaves out any explanatory text. Useful if the
    link is used in a script.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy init

```
NAME
  sy-init - intialize local directory and create remote Syncany repository

SYNOPSIS
  sy init [-P | --plugin=<plugin>] [-o | --plugin-option=<key=value>]
          [-E | --no-encryption] [-G | --no-compression] [-s | --short]
          [-t | --create-target] [-a | --advanced] [-n | --add-daemon]
          [--password]

DESCRIPTION
  This command creates a new remote repository using the specified plugin, and
  initializes the local directory. Unless -o is set, the command is
  interactive and queries the user for input.

  Depending on the chosen plugin chosen (with -P or interactively), different
  plugin-specific options are required or optional.

  Once the 'init' command was successfully executed, the initialized local
  folder can be synced with the newly created repository. The commands
  'up', 'down', 'watch', etc. can be used. Other clients can then be connected
  using the 'connect' command.

OPTIONS
  -P, --plugin=<plugin>
    Selects a plugin to use for the repository. Local files will be synced via
    the storage specified by this plugin. Any of the following available
    plugins can be used: %PLUGINS%

  -o, --plugin-option=<key=value> (multiple options possible)
    Sets a plugin-specific setting in the form of a key/value pair. Each
    plugin defines different mandatory and optional settings. At least, all
    mandatory settings must be specified by this option. All mandatory and
    optional settings can be listed using the 'plugin' command.

  -E, --no-encryption
```

```
    DO NOT USE THIS OPTION. Turns off the encryption for the newly created
    remote repository. All files are stored in plaintext. No password is
    needed for either syncany:// link, multichunk or metadata.

  -G, --no-compression
    Turns off Gzip compression for the newly created remote repository. All
    files are stored in uncompressed form. Can increase indexing performance,
    but will also increase transfer times and remote storage space.

  -t, --create-target
    If not existent, creates the target path on the remote storage. If this
    option is not given, the command will fail if the target folder/path does
    not exist.

  -a, --advanced
    Runs the interactive setup in an advanced mode, querying the user for more
    detailed encryption options. In particular, it is possible to select the
    available symmetric ciphers and modes of operation to encrypt the
    repository with.

  -n, --add-daemon
    The initialized local folder is automatically added to the daemon
    configuration for automatic synchronization if this option is used.

  -s, --short
    The syncany:// link printed after the initialization will be shortened
    using the Syncany link shortener service. This option stores the encrypted
    link on the Syncany servers. The option does not work if -E is enabled.

  --password=<password>
    DO NOT USE THIS OPTION. Set the password used to encrypt the repository.
    This option shouldn't be used, because the password might be visible to
    other users or be stored in history files.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy log

```
NAME
  sy-log - shows recent changes

SYNOPSIS
  sy log [-x | --exclude-empty] [-n | --database-count=<count>]
         [-s | --database-start=<index>] [-f | --file-count=<count>]

DESCRIPTION
  This command displays the recent changes to the repository, grouped
  by the corresponding database versions.

  By default, the command will display the last 10 database versions
  and their associated files. This default value can be changed by
  the -n parameter. The per-database file count can be changed with the
  -f option (default is 100). To hide potentially empty database versions,
  the -x option can be used.

OPTIONS
  -n, --database-count=<count>
    Adjusts the max. number of database versions to be returned and
    displayed by this command. If this option is not set, max. 10
    database versions are displayed. To return all database versions,
    set this option to -1.

  -s, --database-start=<index>
    Adjusts the start index of the databases to return. In combination
    with -n, this option can be used for pagination. If -s is not given,
    the first -n databases will be returned. The default for -s is 0.

  -f, --file-count=<count>
    Adjusts the max. number of changed files per database version to be
    returned and displayed by this command. If this option is not set,
    max. 100 files are displayed. To return all files per database version,
    set this option to -1.

  -x, --exclude-empty
    Excludes empty database version from the result. If this option is
    not given, empty databases will be listed as '(empty)'.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy ls-remote

```
NAME
  sy-ls-remote - list changes on remote repository

SYNOPSIS
  sy ls-remote

DESCRIPTION
  This command compares the list of locally known remote databases with the
  remotely available client databases and prints new/unknown files to the
  console.

  This command is used by the 'down' command to detect which remote databases
  to download and compare.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy ls

```
NAME
  sy-ls - lists and filters the current and past file tree
```

```
SYNOPSIS
  sy ls [-V | --versions] [-t | --types=<types>] [-D | --date=<date>]
        [-r | --recursive] [-f | --full-checksums] [-g | --group]
        [-H | --file-history] [-q | --deleted] [<path-expression>]

DESCRIPTION
  This command lists and filters the file tree based on the local database.
  The file tree selection can be performed using the following selection
  criteria:

  (1) Using the <path-expression>, one can select a file pattern (such as
  `*.txt`) or sub tree (such as `subfolder/`, only with -r). (2) Using -r,
  the command does not only list the folder relative to the
  <path-expression>, but to all sub trees of it. (3)  The -t option limits
  the result set to a certain file type ('f' for files, 'd' for directories,
  and 's' for symlinks). Types can be combined, e.g. `sy ls -tfs` selects
  files and symlinks. (4) The -D option selects the date/time at which to
  select the file tree, e.g. `sy ls -D20m` to select the file tree 20 minutes
  ago or `sy ls -D2014-05-02` to select the file tree at May 2. (5) The -H
  option can be used to select a specific file only. If the option is given,
  the <path-expression> is interpreted as a file history identifier. (6) The
  -q flag will display files that have been deleted from the file system.

  Using the --versions flag, the command also displays the entire version
  history for the selected files. Using --group, the result can be grouped by
  the file history identifier.

OPTIONS
  -V, --versions
    Select and display the entire history of the matching files instead of only
    the last version. Useful with --group.

  -t, --types=<t|d|s>
    Limits the result set to a certain file type ('f' for files, 'd' for
    directories, and 's' for symlinks). Types can be combined, e.g.
    `sy ls -tfs` selects files and symlinks. Default setting is 'tds'.

  -D, --date=<relative-date|absolute-date>
    Selects the file tree at a certain date. The date can be given as a
    relative date to the current time, or an absolute date in form of a
    timestamp.

    Absolute date format: <yy-MM-dd HH:mm:ss>

    Relative date format: <value><unit>, for which <value> may be any
    floating point number and <unit> may be any of the following: s(econds),
    m(inutes), h(ours), d(ays), w(eeks), mo(nths), y(ears). Units may be
    shortened if they are unique. Examples: 5h30m or 1y1mo2d

  -H, --file-history
    If the option is given, the <path-expression> is interpreted as a file
    history identifier. This option can be used to select one specific
    file history. If -H is given, -V is automatically switched on.

  -r, --recursive
    Not only selects the folder relative to the <path-expression>, but to all
    sub trees of it.
  -q, --deleted
    Also selects files that have been deleted from the file system, but are
    still kept in the database. These files can be restored using the
    `sy restore` command. By default, deleted files are not displayed.

  -g, --group
    Only works with --versions. Displays the file versions grouped by file
    history.

  -f, --full-checksums
    Displays full/long checksums instead of shortened checksums.

  <path-expression>
    Selects a file pattern or sub tree of the database using substring and
    wildcard mechanisms. The expression is applied to the relative slash-
    separated path. The only possible wildcard is * (equivalent: ^).

    If <path-expression> does not contain a wildcard, it is interpreted as
    prefix and extended to `<path-expression>*`. If a wildcard is present, no
    wildcard is appended.

    Note: The Linux shell expands the * wildcard if a matching file is
    present. Either use single quotes (e.g. '*.txt') or use ^ instead.

EXAMPLES
  sy ls -r subfolder/
    Selects all file entries of the current file tree in the folder
    'subfolder/', including for instance 'subfolder/some/other/file.txt'.

  sy ls --recursive --types=fs --date=1h30m '*.txt'
    Selects all files and symlinks in the entire file tree that end with .txt
    and existed one and 30 minutes hour ago.

  sy ls --versions --group --recursive
    Selects and displays all file versions and their file histories.
    This selects the entire database. Use with caution.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy plugin

```
NAME
  sy-plugin - list, install, update and remove Syncany plugins

SYNOPSIS
  sy plugin list [-R | --remote-only] [-L | --local-only] [-s | --snapshots]
                 [-a | --api-endpoint=<url>] [<plugin-id>]

  sy plugin install [-s | --snapshot] [-m | --minimal-output]
                    (<URL> | <file> | <plugin-id>)

  sy plugin update [<plugin-id>]
  sy plugin remove <plugin-id>

DESCRIPTION
  This command performs four different actions:

  - It lists the locally installed and remotely available plugins, including
  version information and whether plugins can be upgraded. The 'list' action
  connects to the Syncany host to retrieve remote plugin information. By
  default, only plugin releases will be listed in the result. If instead
  daily snapshots are desired, the -s option can be used.

  - It installs new plugins from either a given URL or a local file. URL and
  local file must point to a valid Syncany plugin JAR file to be installable.
  If <plugin-id> is given, the command will connect to the Syncany host and
  download the desired plugin from there. If instead of the release version
  the daily snapshot shall be installed, the -s option can be used.

  Plugins installed by the 'install' action will be copied to the Syncany
  user directory. On Unix-based systems, this directory is located at
  ~/.config/syncany/plugins, and on Windows at %AppData%\Syncany\plugins.

  - It updates plugins by removing and re-installing them from the Syncany
  host (if they are updatable). Plugins are updatable if they are out-of-date
  and are not 'Global' plugins.

  - It removes locally installed plugins from the user's local plugin
  directory. Only plugins installed by the 'install' action can be removed.
  The plugins shipped with Syncany (e.g. the 'local' plugin) cannot be
  removed.

ACIONS
  The following actions are available within the 'plugin' command:

  list [<args>] [<plugin-id>]
    Lists locally installed plugins and/or remotely available plugins
    on api.syncany.org. If <plugin-id> is given, the result list will
    be shortened to the selected plugin.

    -R, --remote-only
      Turns off local plugin discovery. In particular, the result list will
      not include any information about the locally installed plugins.
      Instead only remotely available plugins will be listed. Cannot be used
      in combination with -L.

    -L, --local-only
      Turns off remote plugin discovery. Contrary to -R, the result list will
      only include information about the locally installed plugins, and no
      information about remote plugins. The Syncany host will not be queried.
      Cannot be used in combination with -R.

    -s, --snapshots
      Instead of listing only plugin release versions (default), the result
      list will also include daily snapshots (if newer snapshots exist).

    -a, --api-endpoint=<url>
      Selects the API endpoint to query for remote plugins. If not given,
      the default endpoint URL will be used (https://api.syncany.org/v3).
  install [<args>] (<URL> | <file> | <plugin-id>)
    Installs a plugin from an arbitrary URL, local file or from the
    available plugins on api.syncany.org (with a plugin identifier).

    -s, --snapshot
      Installs the daily snapshot instead of the release version. Only if
      <plugin-id> is given. Not for <URL> or <file>.

    -m, --minimal-output
      Reduces the output of the command to "OK" or "NOK" instead of reporting
      detailed progress and download URLs.

  update [<plugin-id>]
    Updates the plugin with the plugin identifier <plugin-id> or updates all
    updatable plugins if no identifier is given. Plugins can only be updated
    if they are newer than the installed version and if they are 'User'
    plugins (and not 'Global' plugins). Plugins will be downloaded from
    api.syncany.org.

  remove <plugin-id>
    Uninstalls a plugin entirely (removes the JAR file). This action can
    only be used for plugins that were installed by the user, and not for
    system-wide plugins.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy restore

```
NAME
  sy-restore - restore old or deleted files

SYNOPSIS
  sy restore [-r | --revision=<revision>] [-t | --target=<filename>]
             <file-identifier>

DESCRIPTION
  This command restores old or deleted files from the remote storage.

  As long as a file is known to the local database and the corresponding
  chunks are available on the remote storage, it can be restored using this
  command. The command downloads the required chunks and assembles the file.

  If no target revision is given with -r, the last version is restored. To
  select a revision to restore, the `sy ls` command can be used.

OPTIONS
  -r, --revision=<revision>
    Selects a certain revision/version to restore. If no revision is given,
    the last revision is used.

  -t, --target=<file>
    Defines the target output filename to restore the file to. If this option
    is not given, the default filename is the filename of the restored file
    version, appended with a "restored" suffix. All folders given in the
    target filename will be created.

  <file-identifier>
   Identifier of the file history as printed by the `sy ls` command. The
   file identifier and a revision/version number uniquely identify a single
   version of a file at a certain point in time. The identifier can be
   abbreviated if it is unique in the database.

EXAMPLES
  sy restore 3168ab663e
    Restores the last version of the file with identifier 3168ab663e. If the
    database knows three versions of this file, the second file will be
    restored. Assuming that the original filename was 'folder/file.txt', the
    target filename will be 'folder/file (restored).txt'. If 'folder' does not
    exist, it will be created.

  sy restore --revision=1 --target=restored-file.txt 3168ab663e
    Restores version 1 of the file with the identifier 3168ab663e to the
    target file 'restored-file.txt'. If this file exists, an error will be
    thrown.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy status

```
NAME
  sy-status - list new and changed files in local Syncany folder

SYNOPSIS
  sy status [-f | --force-checksum] [-D | --no-delete]

DESCRIPTION
  This command compares the local file tree on the disk with the local
  database and detects local changes. These changes are printed to the
  console.

  Local changes are detected using the last modified date and the file size
  of a file. If they match the local database, the command assumes that the
  content has not changed (no checksum comparison). If -f is enabled, the
  checksum is additionally compared.

  This command is used by the 'up' command to detect local changes.

OPTIONS
  -f, --force-checksum
    Enforces this command to compare files by checksum, not by file size
    and last modified date only. This option is particularly useful if
    files are modified in-place very often (last modified date and size
    do not change). For large local folders, this option can tremendously
    decrease the performance of this command and increase I/O significantly.
  -D, --no-delete
    With this option, this command will not list locally deleted files. If
    used with the 'up' command, these changes will not be uploaded.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy up

```
NAME
  sy-up - uploads changes in local Syncany folder to remote repository

SYNOPSIS
  sy up [-R | --no-resume] [<status-options>]

DESCRIPTION
  This command detects changes in the local folder, indexes new files and
  uploads changes to the remote repository. If there are local changes, the
  command determines what has changed, packages these changes in new
  multichunks, and uploads them to the remote storage alongside with a delta
  metadata database.

  To determine the local changes, the 'status' command is used. All options
  of the 'status' command can also be used in this command.

  If there are no local changes, the 'up' command will not upload anything -
  no multichunks and no metadata.

  If this command is interrupted during the upload phase, it will try to resume
  the upload unless -R is given. An interrupted upload can only be resumed if
  the last 'up' failed and no 'down' or 'cleanup' has been done since then.

OPTIONS
  -R, --no-resume
    With this option, 'up' will not attempt to resume a locally stored
    transaction. Without this option, an interrupted upload will be resumed.

  All arguments of the 'status' command can be used.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy update

```
NAME
  sy-update - Manages application updates

SYNOPSIS
  sy update check [-s | --snapshots] [-a | --api-endpoint=<url>]
```

```
DESCRIPTION
  This command manages updates of the application. It currently only
  performs update checks, but will likely be extended to automatically
  update the application. The following actions exist:

  - The 'check' action checks if a new application version is available.
  It queries the Syncany API and outputs whether the local copy of the
  application is up-to-date. If it is not, it outputs the newest version
  and a download URL.

ACIONS
  check [<args>]
    Checks with the Syncany API (api.syncany.org) for a new version.

    -s, --snapshots
      Instead of checking against application release versions (default),
      the command will also include daily snapshots.

    -a, --api-endpoint=<url>
      Selects the API endpoint to query against. If not given, the
      default endpoint URL will be used (https://api.syncany.org/v3).

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

## sy watch

```
NAME
  sy-watch - monitor local Syncany folder and automatically sync changes

SYNOPSIS
  sy watch [-i | --interval=<sec>] [-s | --delay=<sec>] [-W | --no-watcher]
           [-a | --announce=<host>:<port>] [-N | --no-announcements]
           [<status-options> | <up-options> | <down-options>]

DESCRIPTION
  Automatically synchronizes the local folder with the repository. The
  command performs the up and down command in an interval, watches the
  file system for changes and subscribes to the Syncany pub/sub server.

  In the default configuration (no options), the command subscribes to the
  Syncany pub/sub server and registers local file system watches in the
  locally synced folder (and all of its subfolders). When local events are
  registered, the command waits a few seconds (waiting for settlement) and
  then triggers the 'up' command. After the upload has finished, a message
  is published to the pub/sub server, telling other clients of this repo
  that there is new data. Clients subscribed to the repository's channel
  will receive this notification and immediately perform a 'down' command.
  This mechanism allows instant synchronization among clients even if a dumb
  storage server (such as FTP) is used.

  In case file system events or pub/sub notifications are missed, the
  periodic synchronization using the 'down' and 'up' command is implemented
  as a fallback.
  Note: The messages exchanged through the pub/sub server do not include any
  confidential data. They only include the repository identifier (randomly
  generated in the 'init' phase), and a client identifier (randomly generated
  on every restart).

OPTIONS
  -s, --delay=<sec>
    Waits for <sec> seconds for file system watcher to settle before starting
    to index newly added files. If many file system actions are executed (e.g.
    copying a large folder), waiting a few seconds ensures that actions
    belonging together are uploaded in a single new database version.
    Default value is 3 seconds.

  -W, --no-watcher
    Disables folder watcher entirely. Local changes in the synced folder (and
    its subfolders) will not be registered right away. Instead, local changes
    will only be detected by the periodic synchronization loop. Unless other
    clients have also set this option, changes by other clients will still be
    detected through the pub/sub server. If -W is set, the -i/--interval
    option becomes more relevant as local synchronization entirely relies on
    the interval.

  -a, --announce=<host>:<port>
    Defines the hostname and the port of the pub/sub server. The pub/sub
    server is used to notify other clients if the local client uploaded new
    data, and to get notified if other clients did so. Default is the central
    Syncany pub/sub server at notify.syncany.org:8080. The SparkleShare
    pub/sub server can be used interchangeably. To set up your own server,
    install a fanout instance from https://github.com/travisghansen/fanout/.

  -N, --no-announcements
    Disables the pub/sub server entirely. Syncany will not connect to the
    server and changes that are published to the pub/sub server are not
    detected. Instead, remote changes will only be detected by the periodic
    synchronization loop. If -N is set, the -i/--interval option becomes more
    relevant as remote synchronization entirely relies on the interval.

  -i, --interval=<sec>
    Sets the synchronization interval of the periodic 'down'/'up' loop to be
    run every <sec> seconds. The sync loop is a fallback only and is not
    relevant if the pub/sub server and the file system watching is enabled.
    Default value is 120 seconds.

  In addition to these options, all arguments of the commands 'status',
  'ls-remote', 'up' and 'down' can be used.

COPYRIGHT
  Syncany 0.4.4-alpha, Distributed under GPLv3,
  Copyright (c) 2011-2015 Philipp C. Heckel
```

# CHAPTER 11

# Appendix B: Videos & Links

There is quite a bit of reading material on Syncany already. Check out the following links and videos. Disclaimer: Some of the videos and articles might be outdated. Syncany moves really quickly.

**Contents**

## Videos

The following lists some random videos we made to demonstrate Syncany. We are developers, not directors or video editors – so apologies for the quality and the lack of spoken word. The two main **video site profiles** are the YouTube channel and the asciinema profile.

- Bash completion (Mar '15, part of this blog post)
- Recent changes & history browser (Jan '15, 36 sec)
- History browser keeps the state; browse previous versions (Jan '15, 36 sec)
- History browser sneak peek (Jan '15, 34 sec)
- SFTP syncing, toggle notifications, install plugins, recent changes (Jan '15, 3 min)
- Init, connect and sync via graphical user interface (Dec '14, 3 min)
- Flickr-based storage plugin in action (Dec '14, 2 min)
- Connect to WebDAV with password via GUI (Dec '14, 1 min)
- GUI Demo: Adding existing/new folders with the Wizard (Dec '14, 2 min)

- Windows/Linux with GUI / tray icons and conflicts (Oct '14, 3 min)
- Progress bar in command line (Sep '14, 1 min)
- Syncany Web Interface Preview (now outdated!) (Jun '14, 2 min)
- Trying out the newest Syncany version with Docker (May '14, 1 min)
- Automatically sync folders (using the S3 plugin) (May '14, 3 min)
- Installing and using plugins (in this video: SFTP) (May '14, 1 min)
- Installing and using plugins (in this video: WebDAV) (May '14, 1 min)
- Automatically sync folders (using the FTP plugin) (Apr '14, 1 min)
- Connect to an existing repository and sync a few files (using the FTP plugin) (Apr '14, 1 min)
- Create a new repository and link the local folder (using the FTP plugin) (Apr '14, 1 min)
- Setup Amazon S3 for two users, and sync two clients with Syncany (Oct '13, 9 min)
- Conflict handling on Linux, using local plugin (Oct '13, 2 min)
- Checkout code, compile and run two clients on Linux, using FTP plugin (Oct '13, 14 min)

## Links

There is quite a bit of reading material on Syncany already. Check out the following links:

**Posts and papers**:

- Syncany: The road to beta – Delft Students On Software Architecture (May 2015)
- Blog post: Syncany explained: idea, progress, development and future (part 1) (Oct 2013)
- Blog post: Deep into the code of Syncany – CLI, application flow and data model (part 2) (Feb 2014)
- Master's thesis: Minimizing remote storage usage and synchronization time using deduplication and multi-chunking: Syncany as an example (2011)

**Diagrams**:

- Diagram: Syncany application flow example
- Diagram: Chunking framework class diagram
- Diagram: Storage plugins class diagram
- Diagram: Database class diagram
- Diagram: Cryptography concept

**Generated JavaDoc and JUnit Reports**:

The up-to-date JavaDoc of the master branch is always compiled to docs.syncany.org/javadoc. It includes the JavaDoc of all Gradle modules in the repo. All results of the JUnit tests and Cobertura coverage reports are compiled to reports.syncany.org.