
Syncthing Documentation

Release v0.11

The Syncthing Authors

Jan 30, 2022

INTRODUCTION

1 Contact

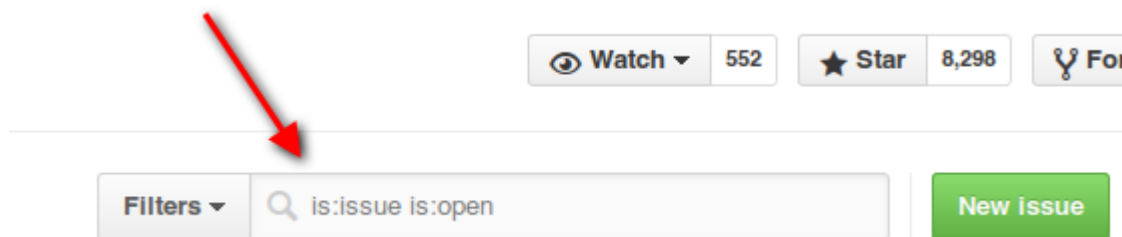
3

As a new user, the *[getting started guide](#)* is a good place to start, then perhaps moving on to *[the FAQ](#)*. If you run into trouble getting devices to connect to each other, the page about *[firewall setup](#)* explains the networking necessary to get it to work.

As a developer looking to get started with a contribution, see *[how to build](#)*, *[how to debug](#)* and the *[contribution guidelines](#)*. This documentation site can be edited on *[Github](#)*.

CONTACT

- If you're looking for specific people to talk to, check out the [Project Presentation](#).
- To report bugs or request features, please use the [issue tracker](#). Before you do so, make sure you are running the [latest version](#), and please do a quick search to see if the issue has already been reported.



- To report security issues, please follow the instructions on the [Security](#) page.
- To get help and support, to discuss scenarios, or just connect with other users and developers you can head over to the [friendly forum](#).
- For a more real time experience, there's also an IRC channel [#syncthing](#) on [Freenode](#).
- For other concerns you may reach out to members of the core team, currently [@calmh](#), [@AudriusButkevicius](#) and [@Zillode](#).

The main documentation for the site is organized into a couple of sections. You can use the headings in the left sidebar to navigate the site.

1.1 Getting Started

For this guide let's assume you have two machines between which you want to synchronise files. In keeping with Syncthing terminology they are going to be called “devices” in the rest of the documentation. The “local device” is the one you are configuring, viewing status for, etc, while the “remote device” is the other machine.

The best way to follow this guide is to do the install on both machines and configure them in parallel. If both machines aren't accessible simultaneously that's fine, the results will just be a little less immediately obvious.

A film version of this transcript is available on YouTube (contributed by [@theincogtion](#)). [This video](#) shows how to install Syncthing on Ubuntu/Debian/Mint using PPA, also available in [German](#). [This video](#) shows how to install Syncthing on Windows, also available in [German](#)

1.1.1 Installing

We suggest you have a look to many of the *Community Contributions* which let you pick a flavor of Syncthing that best fits your scenario. For example, if you are interested in a cross-platform GUI application you can check out *Syncthing-GTK*. The community has also developed Windows, Android and many more specific flavors that help you run Syncthing on your devices. Currently all community flavors run the same Syncthing core underneath, so don't worry about changing your flavor at a later point in time. The remainder of this page will explain how to set up two devices with the core Syncthing flavor.

Syncthing

Grab the [latest release](#) of Syncthing for your operating system and unpack it. There will be a binary called `syncthing` (or `syncthing.exe` on Windows). Start this in whatever way you are most comfortable with; double-clicking should work in any graphical environment, but I'll use the terminal to better illustrate what happens. At first start, Syncthing will generate a configuration and some keys and then start the admin GUI in your browser. Something like the following will be printed in the terminal:

```
$ syncthing
[monitor] 15:56:58 INFO: Starting syncthing
15:56:58 INFO: Generating RSA key and certificate for syncthing...
[ANSMX] 15:57:05 INFO: syncthing v0.10.14 (go1.4 darwin-amd64 default) jb@syno...
[ANSMX] 15:57:05 INFO: My ID: ANSMXYD-E6CF3JC-TCVPYGF-GXJPHSJ-MKUXBUQ-ZSPOKXH-...
[ANSMX] 15:57:05 INFO: No config file; starting with empty defaults
[ANSMX] 15:57:05 INFO: Edit gsl/config.xml to taste or use the GUI
[ANSMX] 15:57:05 INFO: Starting web GUI on http://127.0.0.1:8384/
[ANSMX] 15:57:05 INFO: Loading HTTPS certificate: open gsl/https-cert.pem: no ...
[ANSMX] 15:57:05 INFO: Creating new HTTPS certificate
[ANSMX] 15:57:05 INFO: Generating RSA key and certificate for syno...
[ANSMX] 15:57:07 INFO: Starting UPnP discovery...
[ANSMX] 15:57:13 INFO: UPnP discovery complete (found 0 devices).
[ANSMX] 15:57:13 INFO: Starting local discovery announcements
[ANSMX] 15:57:13 INFO: Local discovery over IPv4 unavailable
[ANSMX] 15:57:13 INFO: Starting global discovery announcements
[ANSMX] 15:57:13 OK: Ready to synchronize default (read-write)
[ANSMX] 15:57:13 INFO: Device ANSMXYD-E6CF3JC-TCVPYGF-GXJPHSJ-MKUXBUQ-ZSPOKXH-...
[ANSMX] 15:57:13 INFO: Completed initial scan (rw) of folder default
```

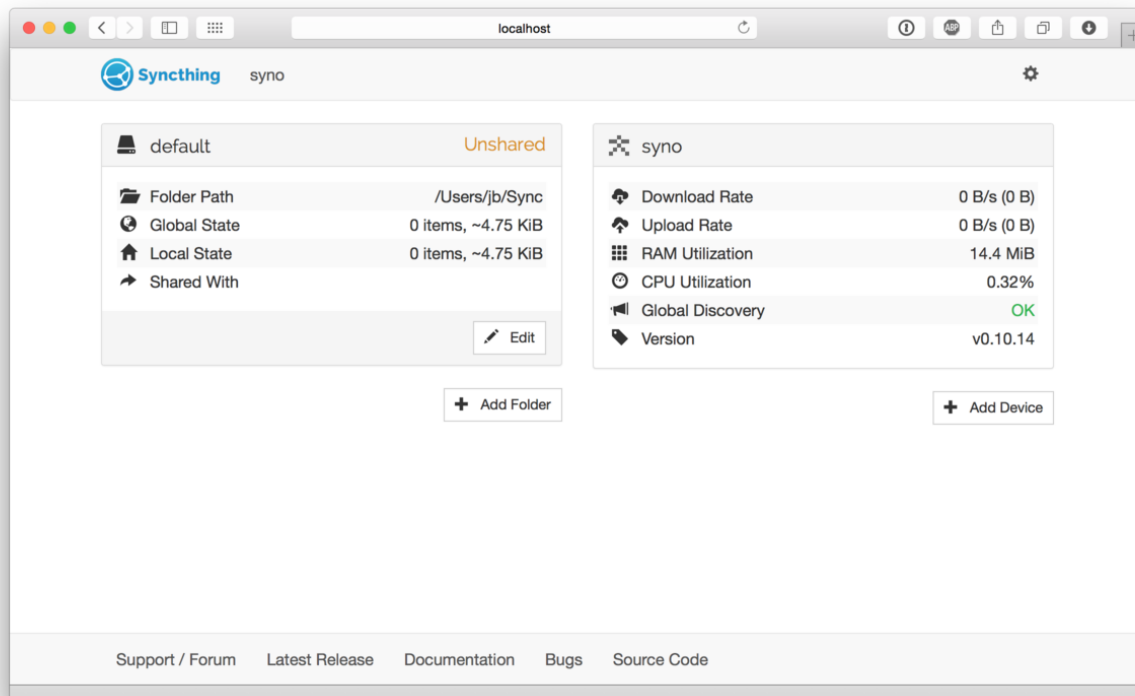
At this point Syncthing will also have set up a folder called `default` for you, in a directory called `Sync` in your home directory. You can use this as a starting point, then remove it or add more folders later.

1.1.2 Configuring

The admin GUI starts automatically and remains available on `http://localhost:8384/`. It should look something like this:

On the left is the list of “folders”, or directories to synchronize. You can see the `default` folder was created for you, and it's currently marked “Unshared” since it's not yet shared with any other device. On the right is the list of devices. Currently there is only one device: the computer you are running this on.

For Syncthing to be able to synchronize files with another device, it must be told about that device. This is accomplished by exchanging “device IDs”. A device ID is a unique, cryptographically-secure identifier that is generated as part of the key generation the first time you start Syncthing. It is printed in the log above, and you can see it in the web GUI by selecting the “gear menu” (top right) and “Show ID”.



Two devices will *only* connect and talk to each other if they both know about the other’s device ID. Since the configuration must be mutual for a connection to happen, device IDs don’t need to be kept secret.

To get your two devices to talk to each other, click “Add Device” at the bottom right on both, and enter the device ID of the other side. You should also select the folder(s) that you want to share with this device. The device name is optional, but you can set it to something that makes sense for you to remember what device this is.

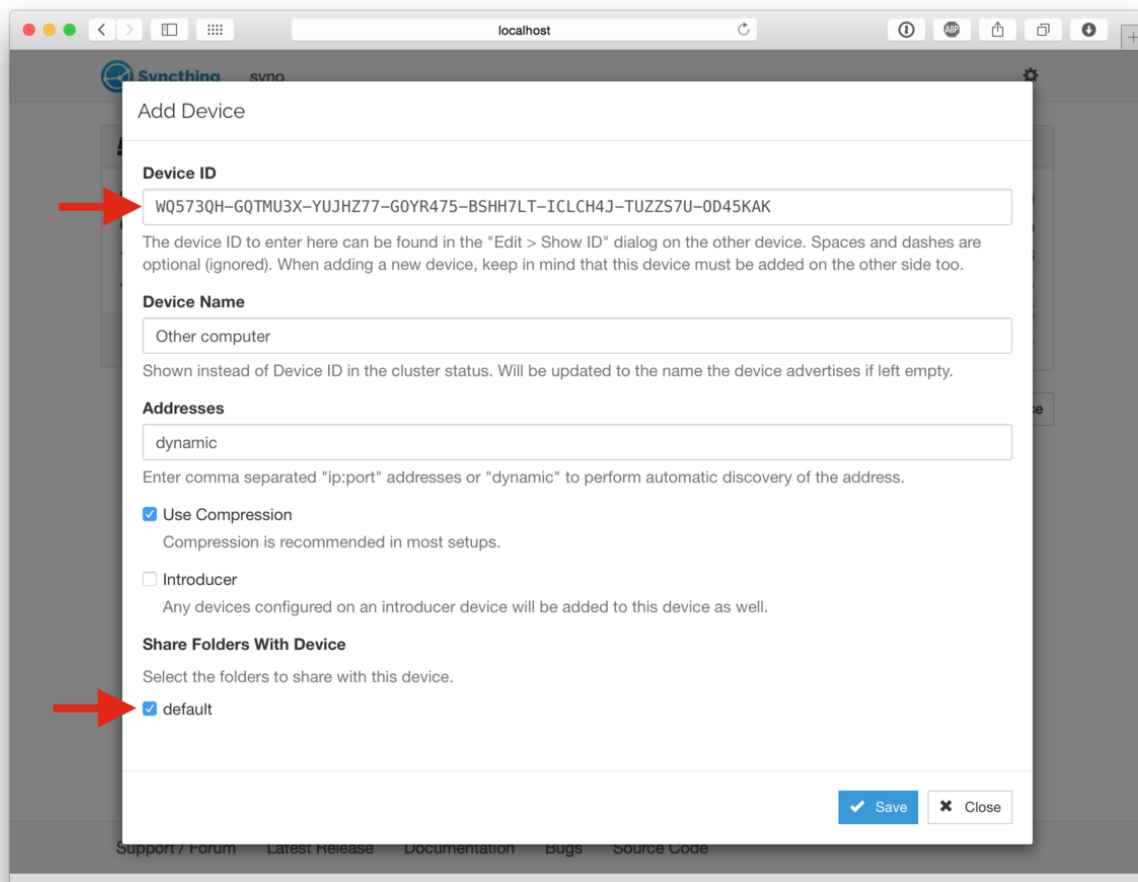
Once you click “Save” the new device will appear on right side of the GUI (although disconnected) and a prompt will be shown to indicate the need for a restart.

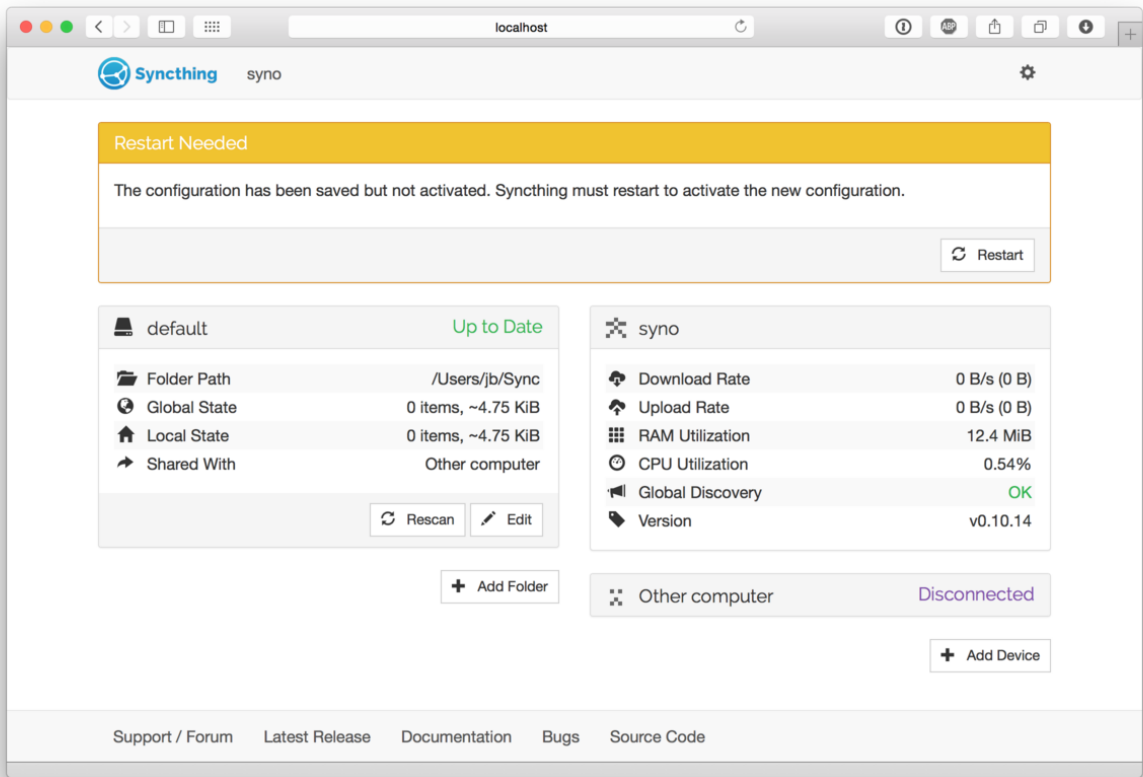
Syncthing needs to be restarted for some configuration changes to take effect (such as sharing folders with new devices). When you click “Restart” Syncthing will first restart:

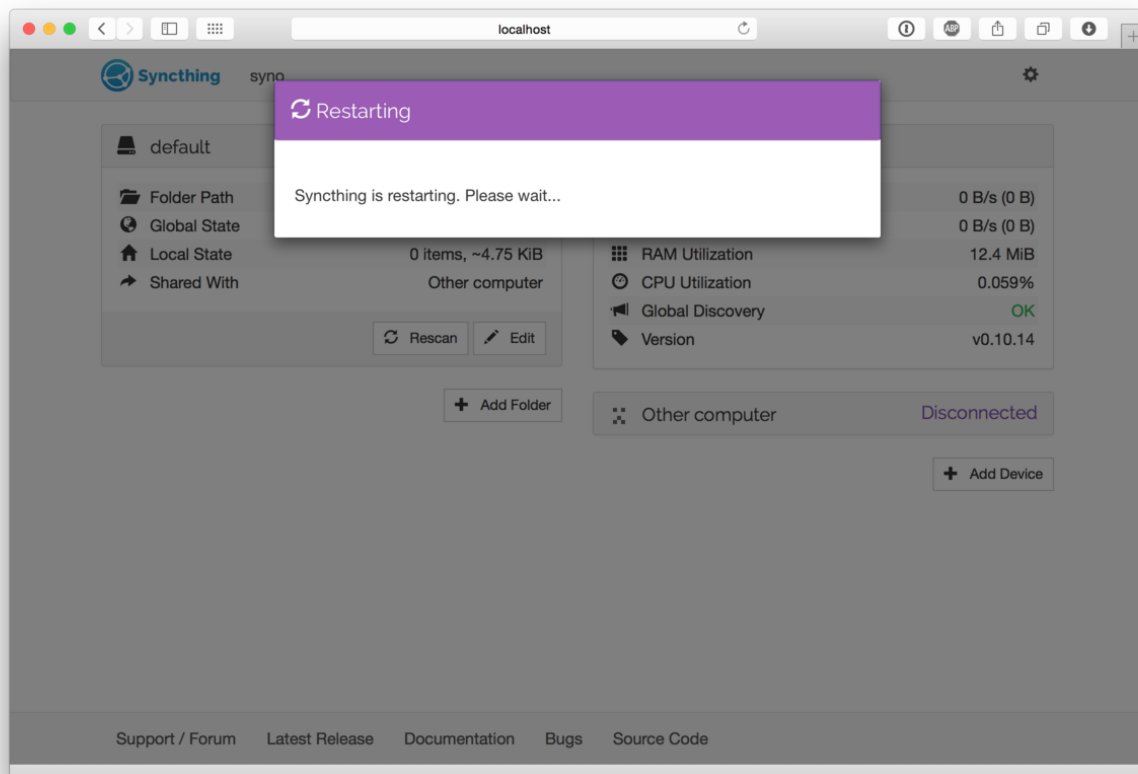
and then come back up and after a while (up to a minute) connect to the other device. Remember though that you need to do the above process on both devices, and only once you’ve done this on both devices will they be able to connect.

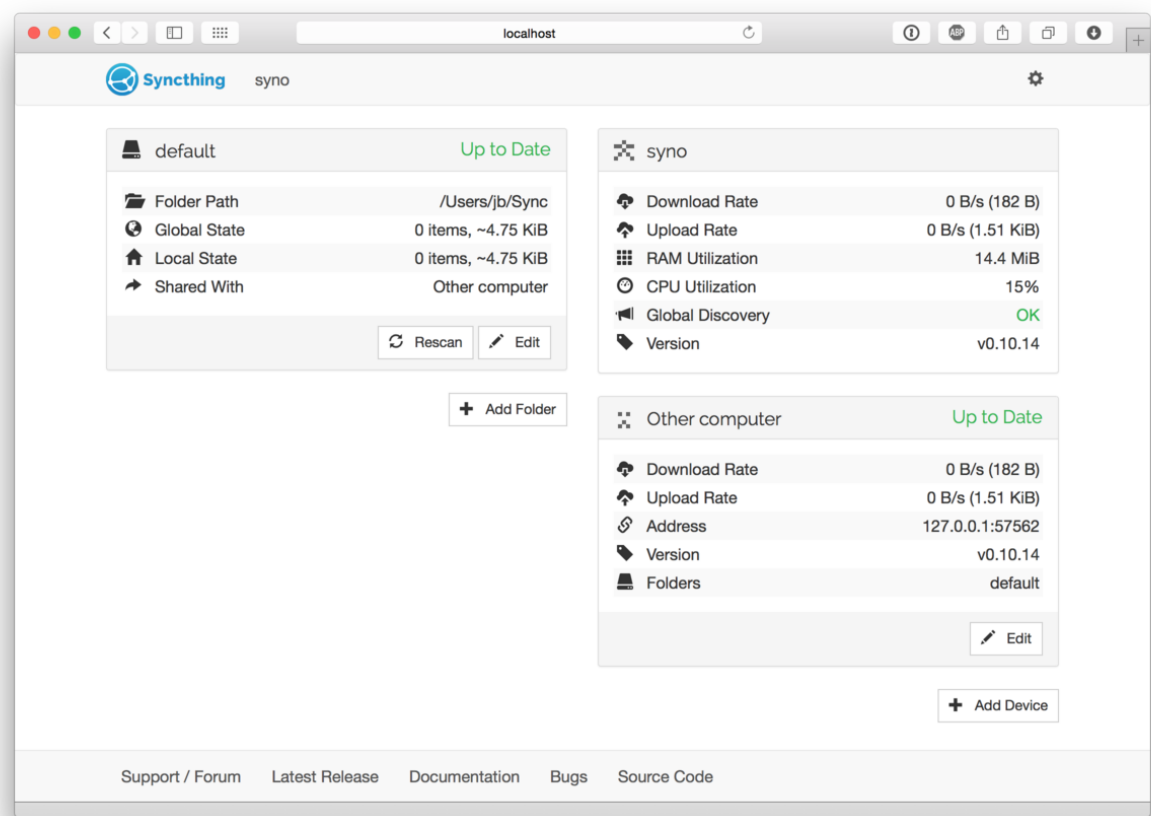
At this point the two devices share an empty directory. Adding files to the Sync directory on either device will synchronize those files to the other side. Each device scans for changes every 60 seconds, so changes can take a little over a minute to propagate to the other side. The rescan interval can be changed for each folder by clicking on a folder, clicking “Edit” and entering a new value for “Rescan Interval”.

Good luck and have fun! There is more [documentation](#) and if you run into trouble feel free to post a question in the [support forum](#). If you have problems getting this to connect, first take a look at [Firewall Setup](#), then look at any error messages in the GUI or on the console and if necessary move on to [Debugging Syncthing](#).





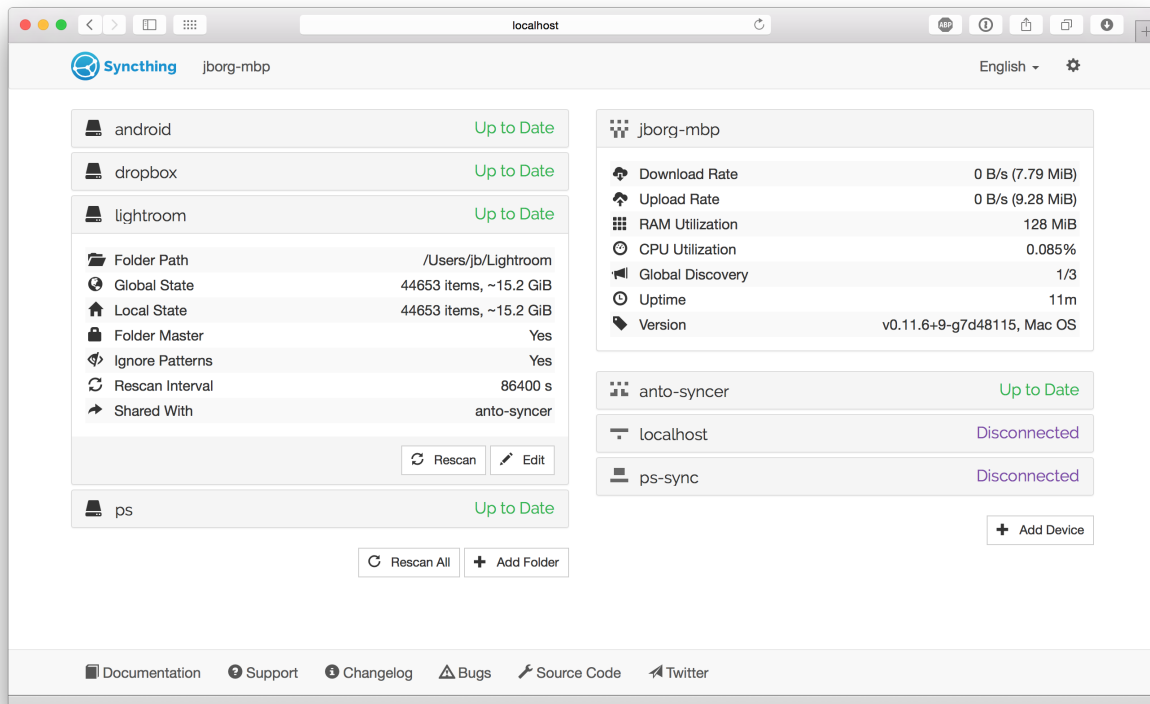




1.2 An intro to the GUI

1.2.1 Folder View

The left side of the screen shows the ID and current state of all configured folders. Clicking the folder name makes that section expand to show more detailed folder information, and buttons for editing the configuration or forcing a rescan.



A folder can be in any one of these states:

- *Unknown* while the GUI is loading.
- *Unshared* when you have not shared this folder,
- *Stopped* when the folder has experienced an error,
- *Scanning* while the folder is looking for local changes,
- *Up to Date* when the folder is in sync with the rest of the cluster,
- *Syncing* when this device is downloading changes from the network.

Among the folder details, you can see the current “Global State” and “Local State” summaries, as well as the amount of “Out of Sync” data if the the folder state is not up to date.

- *Global State* indicates how much data the fully up to date folder contains - this is basically the sum of the newest versions of all files from all connected devices. This is the size of the folder on your computer when it is fully in sync with the cluster.
- *Local State* shows how much data the folder actually contains right now. This can be more or less than the global state, if the folder is currently synchronizing with other devices.

- *Out of Sync* shows how much data that needs to be synchronized from other devices. Note that this is the sum of all out of sync *files* - if you already have parts of such a file, or an older version of the file, less data than this will need to be transferred over the network.

1.2.2 Device View

The right side of the screen shows the overall state of all configured devices. The local device (your computer) is always at the top, with remote devices in alphabetical order below. For each device you see its current state and, when expanded, more detailed information. All transfer rates (“Download Rate” and “Upload Rate”) are from the perspective of your computer, even those shown for remote devices. The rates for your local device is the sum of those for the remote devices.

1.3 Project Presentation

These are the various projects under the Syncthing umbrella and the people chiefly responsible for maintaining them.

1.3.1 Syncthing

Syncthing is the core CLI application, used by the Android and native UI wrappers.

- Jakob Borg / [@calmh](#) (Founder)
- Audrius Butkevicius / [@AudriusButkevicius](#)
- Lode Hoste / [@Zillode](#)

1.3.2 syncthing-android

syncthing-android is the Android packaging and native UI on top of Syncthing.

- Felix Ableitner / [@Nutomic](#) (Founder)
- Lode Hoste / [@Zillode](#)

1.3.3 syncthing-inotify

syncthing-inotify is a filesystem notifications watcher on top of Syncthing.

- Lode Hoste / [@Zillode](#) (Founder)

1.3.4 Syncthing-GTK

Syncthing-GTK is a native UI wrapper on top of Syncthing.

- Tomas Cerveny / [@kozec](#) (Founder)

1.4 FAQ

1.4.1 General

What is Syncthing?

Syncthing is an application that lets you synchronize your files across multiple devices. This means the creation, modification or deletion of files on one machine will automatically be replicated to your other devices. We believe your data is your data alone and you deserve to choose where it is stored. Therefore Syncthing does not upload your data to the cloud but exchanges your data across your machines as soon as they are online at the same time.

Is it “syncthing”, “Syncthing” or “SyncThing”?

It’s **Syncthing**, although the command and source repository is spelled `syncthing` so it may be referred to in that way as well. It’s definitely not SyncThing, even though the abbreviation `st` is used in some circumstances and file names.

How does Syncthing differ from BitTorrent Sync?

The two are different and not related. Syncthing and BitTorrent Sync accomplish some of the same things, namely syncing files between two or more computers.

BitTorrent Sync by BitTorrent, Inc is a proprietary peer-to-peer file synchronization tool available for Windows, Mac, Linux, Android, iOS, Windows Phone, Amazon Kindle Fire and BSD. ¹ Syncthing is an open source file synchronization tool.

Syncthing uses an open and documented protocol, and likewise the security mechanisms in use are well defined and visible in the source code. BitTorrent Sync uses an undocumented, closed protocol with unknown security properties.

1.4.2 Usage

What things are synced?

The following things are *always* synchronized:

- File Contents
- File Modification Times

The following may be synchronized or not, depending:

- File Permissions (When supported by file system. On Windows, only the read only bit is synchronized.)
- Symbolic Links (When supported by the OS. On Windows Vista and up, requires administrator privileges. Links are synced as is and are not followed.)

The following is *not* synchronized;

- File or Directory Owners and Groups (not preserved)
- Directory Modification Times (not preserved)
- Hard Links (followed, not preserved)
- Extended Attributes, Resource Forks (not preserved)
- Windows, POSIX or NFS ACLs (not preserved)

- Devices, FIFOs, and Other Specials (ignored)
- Sparse file sparseness (will become unsparse)

Is synchronization fast?

Syncthing segments files into pieces, called blocks, to transfer data from one device to another. Therefore, multiple devices can share the synchronization load, in a similar way as the torrent protocol. The more devices you have online (and synchronized), the faster an additional device will receive the data because small blocks will be fetched from all devices in parallel.

Syncthing handles renaming files and updating their metadata in an efficient manner. This means that renaming a large file will not cause a retransmission of that file. Additionally, appending data to existing large files should be handled efficiently as well.

Temporary files are used to store partial data downloaded from other devices. They are automatically removed whenever a file transfer has been completed or after the configured amount of time which is set in the configuration file (24 hours by default).

Should I keep my device IDs secret?

No. The IDs are not sensitive. Given a device ID it's possible to find the IP address for that node, if global discovery is enabled on it. Knowing the device ID doesn't help you actually establish a connection to that node or get a list of files, etc.

For a connection to be established, both nodes need to know about the other's device ID. It's not possible (in practice) to forge a device ID. (To forge a device ID you need to create a TLS certificate with that specific SHA-256 hash. If you can do that, you can spoof any TLS certificate. The world is your oyster!)

See also:

[Understanding Device IDs](#)

What if there is a conflict?

Syncthing does recognize conflicts. When a file has been modified on two devices simultaneously, one of the files will be renamed to `<filename>.sync-conflict-<date>-<time>.<ext>`. The device which has the larger value of the first 63 bits for his device ID will have his file marked as the conflicting file. Note that we only create `sync-conflict` files when the actual content differs.

Beware that the `<filename>.sync-conflict-<date>-<time>.<ext>` files are treated as normal files after they are created, so they are propagated between devices. We do this because the conflict is detected and resolved on one device, creating the `sync-conflict` file, but it's just as much of a conflict everywhere else and we don't know which of the conflicting files is the "best" from the user point of view. Moreover, if there's something that automatically causes a conflict on change you'll end up with `sync-conflict-...sync-conflict -...-sync-conflict` files.

How to configure multiple users on a single machine?

Each user should run their own Syncthing instance. Be aware that you might need to configure ports such that they do not overlap (see the `config.xml`).

Is Syncthing my ideal backup application?

No, Syncthing is not a backup application because all changes to your files (modification, deletion, etc) will be propagated to all your devices. You can enable versioning, but we encourage the use of other tools to keep your data safe from your (or our) mistakes.

Why is there no iOS client?

Alternative implementation Syncthing (using the Syncthing protocol) are being developed at this point in time to enable iOS support. Additionally, it seems that the next version of Go will support the darwin-arm architecture such that we can compile the mainstream code for the iOS platform.

Why does it use so much CPU?

1. When new or changed files are detected, or Syncthing starts for the first time, your files are hashed using SHA-256.
2. Data that is sent over the network is first compressed and then encrypted using AES-128. When receiving data, it must be decrypted and decompressed.

Hashing, compression and encryption cost CPU time. Also, using the GUI causes a certain amount of CPU usage. Note however that once things are *in sync* CPU usage should be negligible.

How can I exclude files with brackets ([]) in the name?

The patterns in `.stignore` are glob patterns, where brackets are used to denote character ranges. That is, the pattern `q[abc]x` will match the files `qax`, `qbx` and `qcx`.

To match an actual file *called* `q[abc]x` the pattern needs to “escape” the brackets, like so: `q\[abc\]x`.

Why is the setup more complicated than BTSync?

Security over convenience. In Syncthing you have to setup both sides to connect two nodes. An attacker can’t do much with a stolen node ID, because you have to add the node on the other side too. You have better control where your files are transferred.

How do I access the web GUI from another computer?

The default listening address is `127.0.0.1:8384`, so you can only access the GUI from the same machine. Change the GUI `listen` address through the web UI from `127.0.0.1:8384` to `0.0.0.0:8384` or change the `config.xml`:

```
<gui enabled="true" tls="false">  
  <address>127.0.0.1:8384</address>
```

to

```
<gui enabled="true" tls="false">
  <address>0.0.0.0:8384</address>
```

Then the GUI is accessible from everywhere. You should most likely set a password and enable HTTPS now. You can do this from inside the GUI.

If both your computers are Unixy (Linux, Mac, etc) You can also leave the GUI settings at default and use an ssh port forward to access it. For example,

```
$ ssh -L 9090:127.0.0.1:8384 user@othercomputer.example.com
```

will log you into othercomputer.example.com, and present the *remote* Syncthing GUI on <http://localhost:9090> on your *local* computer. You should not open more than one Syncthing GUI in a single browser due to conflicting X-CSRFTokens. Any modification will be rejected. See [issue #720](#) to work around this limitation.

The CSRF tokens are stored using cookies. Therefore, if you get the message Syncthing seems to be experiencing a problem processing your request, you should verify the cookie settings of your browser.

Why do I see Syncthing twice in task manager?

One process manages the other, to capture logs and manage restarts. This makes it easier to handle upgrades from within Syncthing itself, and also ensures that we get a nice log file to help us narrow down the cause for crashes and other bugs.

Where do Syncthing logs go to?

Syncthing logs to stdout by default. On Windows Syncthing by default also creates `syncthing.log` in Syncthing's home directory (check `-help` to see where that is).

How do I upgrade Syncthing?

- If automatic upgrades is enabled (which is the default), Syncthing will upgrade itself automatically within 24 hours of a new release.
- The upgrade button appears in the web GUI when a new version has been released. Pressing it will perform an upgrade.
- To force an upgrade from the command line, run `syncthing -upgrade`.

Note that your system should have CA certificates installed which allow a secure connection to GitHub (e.g. FreeBSD requires `sudo pkg install ca_root_nss`). If `curl` or `wget` works with normal HTTPS sites, then so should Syncthing.

Where do I find the latest release?

We release new versions through GitHub. The latest release is always found [on the release page](#). Unfortunately GitHub does not provide a single URL to automatically download the latest version. We suggest to use the GitHub API at <https://api.github.com/repos/syncthing/syncthing/releases/latest> and parsing the JSON response.

1.5 Firewall Setup

1.5.1 Port Forwards

If you have a NAT router which supports UPnP, the easiest way to get a working port forward is to make sure UPnP setting is enabled on both Syncthing and the router – Syncthing will try to handle the rest. If it succeeds you will see a message in the console saying:

```
Created UPnP port mapping for external port XXXXX on UPnP device YYYYY.
```

If this is not possible or desirable you should set up a port forward for port **22000/TCP**, or the port set in the *Sync Protocol Listen Address* setting. The external forwarded port and the internal destination port has to be the same (i.e. 22000/TCP).

Communication in Syncthing works both ways. Therefore if you set up port forwards for one device, other devices will be able to connect to it even when they are behind a NAT network or firewall.

1.5.2 Local Firewall

If your PC has a local firewall, you will need to open the following ports for incoming traffic:

- Port **22000/TCP** (or the actual listening port if you have changed the *Sync Protocol Listen Address* setting.)
- Port **21025/UDP** (for discovery broadcasts)

1.5.3 Remote Web GUI

To be able to access the web GUI from other computers, you need to change the *GUI Listen Address* setting from the default 127.0.0.1:8384 to 0.0.0.0:8384. You also need to open the port in your local firewall if you have one.

Tunneling via SSH

If you have SSH access to the machine running Syncthing but would rather not open the web GUI port to the outside world, you can access it through a SSH tunnel instead. You can start a tunnel with a command like the following:

```
ssh -L 9999:localhost:8384 machine
```

This will bind to your local port 9999 and forward all connections from there to port 8384 on the target machine. This still works even if Syncthing is bound to listen on localhost only.

You can forward multiple ports corresponding to many machines this way, but because Syncthing uses session cookies for the entire domain (i.e. your local machine), you will need to connect to each control panel in a separate browser instance or explicitly issue a browser reload when switching between them.

1.6 Starting Syncthing Automatically

Warning: This page may be outdated and requires review.

1.6.1 Windows

There is currently no official installer available for Windows. However, there are a number of easy solutions.

Third-party Tools

There are a number of third-party utilities which aim to address this issue. These typically provide an installer, let Syncthing start automatically, and a more polished user experience (e.g. by behaving as a “proper” Windows application, rather than forcing you to start your browser to interact with Syncthing).

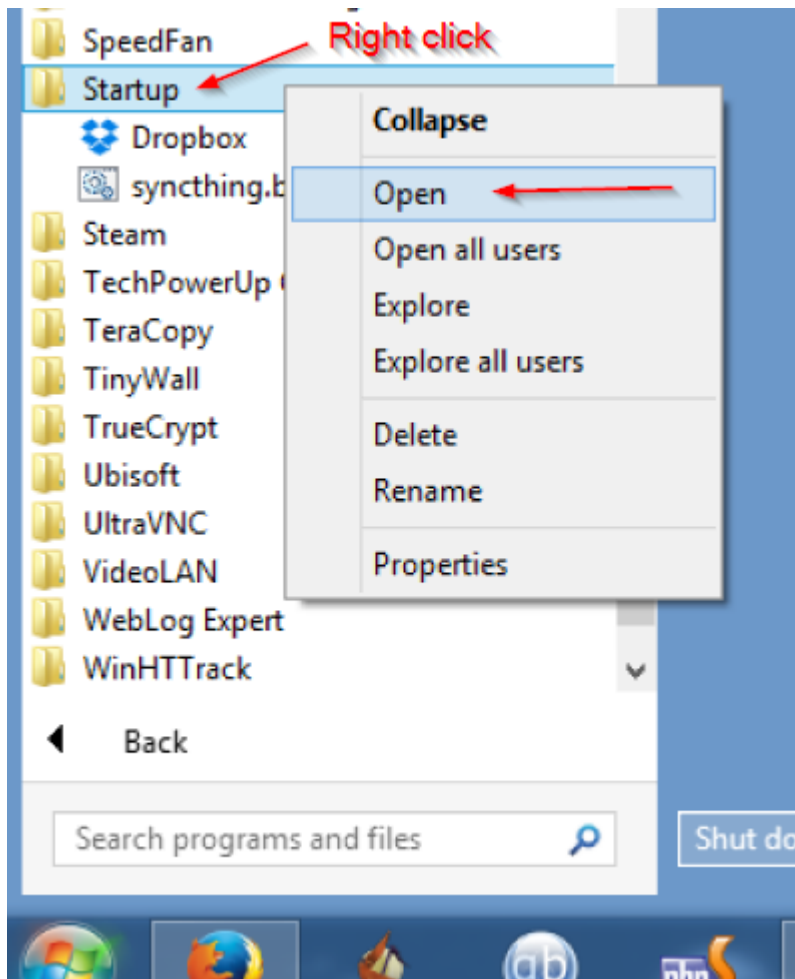
See also:

Windows GUI Wrappers, Cross-platform GUI Wrappers.

Start on Login

Starting Syncthing on login, without a console window or browser opening on start, is relatively easy.

1. Find the correct link of the Windows binary from the [Syncthing website](#) (choose **amd64** if you have a 64-bit version of Windows)
2. Extract the files in the folder (syncthing-windows-*) in the zip to the folder C:\syncthing
3. Go to the C:\syncthing folder, make a file named syncthing.bat
4. Right-click the file and choose **Edit**. The file should open in Notepad or your default text editor.
5. Paste the following command into the file and save the changes: `start "Syncthing" syncthing.exe -no-console -no-browser`
6. Right-click on syncthing.bat and press “Create Shortcut”
7. Right-click the shortcut file syncthing.bat - **Shortcut** and click **Copy**
8. Click **Start**, click **All Programs**, then click **Startup**. Right-click on **Startup** then click **Open**.



9. Paste the shortcut (right-click in the folder and choose **Paste**, or press CTRL+V)

Syncthing will now automatically start the next time Windows boots. No console or browser window will pop-up. Access the interface by browsing to <http://localhost:8384/>

If you prefer slower indexing but a more responsive system during scans, copy the following command instead of the command in step 5:

```
start "Syncthing" /low syncthing.exe -no-console -no-browser
```

Run independent of user login

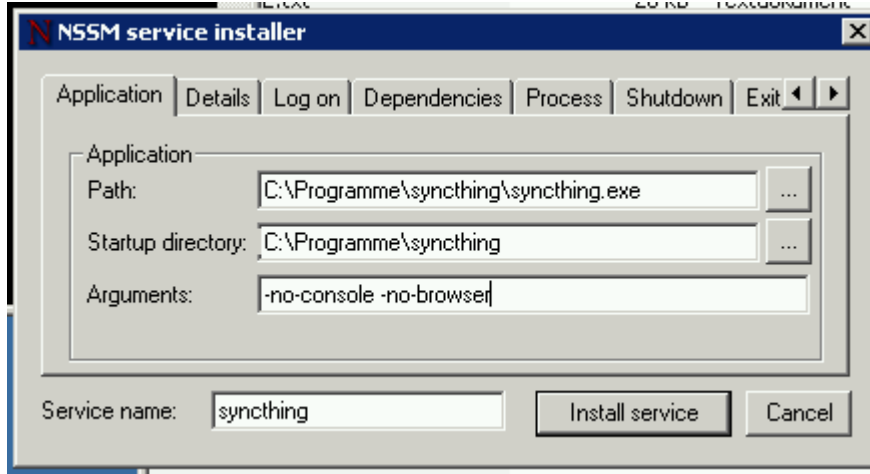
Warning: There are important security considerations with this approach. If you do not secure Syncthing's GUI (and REST API), then **any** process running with **any** permissions can read/write **any** file on your filesystem, by opening a connection with Syncthing.

Therefore, you **must** ensure that you set a GUI password, or run Syncthing as an unprivileged user.

With the above configuration, Syncthing only starts when a user logs in onto the machine. This is not optimal on servers, where a machine can run long times after a reboot without anyone logged in. In this case, it is best to create a service that runs as soon as Windows starts. This can be achieved using nssm.

Note that starting Syncthing on login is the preferred approach for almost any end-user scenario. The only scenario where running Syncthing as a service makes sense is for (mostly) headless servers, administered by a sysadmin who knows enough to understand the security implications.

1. Download and extract `nssm` to a folder where it can stay (e.g. `c:Files` or the Syncthing folder).
2. run `nssm.exe install syncthing`
3. Select `syncthing.exe` in the first tab and enter `-no-console -no-browser` as Arguments



4. at the Details tab you can switch to *Automatic (Delayed Start)* to start it only some time after boot and speed up the boot process (optional)
5. At the *Log On* tab you can enter a username and password for the user to run Syncthing as. This user needs to have access to all the synced folders. Usually, you can leave it as the System account.
6. At the Process Tab you can change the priority to low if you want a more responsive system at the cost of longer sync time
7. Click the *Install Service* Button
8. Start the service using the windows service manager, enter `sc start syncthing` in a console window or restart the PC.
9. Connect to the Syncthing UI, enable HTTPS, and set a secure username and password.

1.6.2 Mac OS X

Using homebrew

1. `brew install syncthing`
2. Follow the info to autostart Syncthing using `launchctl`. At the moment this is done using this command:
`launchctl load ~/Library/LaunchAgents/homebrew.mxcl.syncthing.plist.`

Without homebrew

Download Syncthing for Mac: <https://github.com/syncthing/syncthing/releases/latest>.

1. Copy the syncthing binary (the file you would open to launch Syncthing) in a directory called bin in your home directory. If “bin” does not exist, create it.
2. Edit the `syncthing.plist` (located in `/etc/macosex-launchd`) in the two places that refer to your home directory; that is, replace `/Users/jb` with your actual home directory location.
3. Copy the `syncthing.plist` file to `~/Library/LaunchAgents`. If you have trouble finding this location select the “Go” menu in Finder and choose “Go to folder...” and then type `~/Library/LaunchAgents`. Copying to `~/Library/LaunchAgents` will require admin password in most cases.
4. Log out and back in again. Or, if you do not want to log out, you can run this command in terminal: `launchctl load ~/Library/LaunchAgents/syncthing.plist`

Note: You probably want to turn off “Start Browser” in the web GUI settings to avoid it opening a browser window on each login. Then, to access the GUI type `127.0.0.1:8384` (by default) into Safari.

1.6.3 Linux

Ubuntu like systems

1. Click the dashboard (hit ‘Win’ button).
2. Open ‘Startup Applications’.
3. Click ‘Add’.
4. Fill out the form:
 - Name: Syncthing
 - Command: `/path/to/syncthing/binary -no-browser -home="/home/your_user/.config/syncthing"`

Supervisord

Add following to your `/etc/supervisord.conf`:

```
[program:syncthing]
command = /path/to/syncthing/binary -no-browser -home="/home/some_user/.config/syncthing"
directory = /home/some_user/
autorestart = True
user = some_user
environment = STNORESTART="1"
```


systemd

systemd is a suite of system management daemons, libraries, and utilities designed as a central management and configuration platform for the Linux computer operating system. It also offers users the ability to manage services under the user's control with a per-user systemd instance, enabling users to start, stop, enable, and disable their own units. Service files for system are provided by Syncthing and can be found in [etc/linux-systemd](#). Several distros (including arch linux) ship these service files with the Syncthing package. If your distro provides a systemd service file for Syncthing you can skip step 2.

How to use the system instance

Running Syncthing as a system service ensures that Syncthing is run at startup even if the Syncthing user has no active session. Since the system service keeps Syncthing running even without an active user session, it is intended to be used on a *server*.

1. Create the user who should run the service, or choose an existing one.
2. Copy the `system/syncthing@.service` file into the [load path of the system instance](#).
3. Enable and start the service. Append the Syncthing user after the @:

```
systemctl enable syncthing@myuser.service
systemctl start syncthing@myuser.service
```

How to use the user instance

Running Syncthing as a user service ensures that Syncthing is run after the Syncthing user has created a session (e.g. via the graphical login screen or ssh). Thus, the user service is intended to be used on a (*multiuser*) *desktop computer*. It avoids unnecessarily running Syncthing instances.

1. Create the user who should run the service, or choose an existing one.
2. Copy the `user/syncthing.service` file into the [load path of the user instance](#). To do this without root privileges you can use `~/.config/systemd/user/`.
3. Enable and start the service:

```
systemctl --user enable syncthing.service
systemctl --user start syncthing.service
```

To check if Syncthing runs properly you can use the status subcommand:

```
systemctl status syncthing@myuser.service
systemctl --user status syncthing.service
```

Using the journal

Systemd logs everything into the journal. You can easily access Syncthing log messages (`-e` lets the pager jump to the very end):

```
journalctl -e -u syncthing@myuser.service
journalctl -e --user-unit=syncthing.service
```

Debugging

If you are asked on the bugtracker to start Syncthing with specific environment variables it will not work the easy way. Systemd isolates each service and it cannot access global environment variables. The solution is to add this variables to the service file instead. Just use:

```
systemctl edit syncthing@myuser.service
systemctl --user edit syncthing.service
```

This will create an additional configuration file automatically and you can define (or overwrite) further service parameters like e.g. `Environment=STTRACE=model`.

1.7 Syncthing Configuration

Warning: This page may be outdated and requires review. Attributes have been added that are not documented.

1.7.1 Synopsis

```
$HOME/.config/syncthing/config.xml
$HOME/Library/Application Support/Syncthing
%AppData%/Syncthing
%localappdata%/Syncthing
```

1.7.2 Description

Syncthing uses a single directory to store configuration, crypto keys and index caches. The location defaults to `$HOME/.config/syncthing` (Unix-like), `$HOME/Library/Application Support/Syncthing` (Mac), `%AppData%/Syncthing` (Windows XP) or `%localappdata%/Syncthing` (Windows 7/8). It can be changed at runtime using the `-home` flag. In this directory the following files are located:

cert.pem The device's RSA public key, named "cert" for legacy reasons.

key.pem The device's RSA private key. This needs to be protected.

config.xml The configuration file, in XML format.

https-cert.pem The certificate for HTTPS GUI connections.

https-key.pem The key for HTTPS GUI connections.

index/ A directory holding the database with metadata and hashes of the files currently on disk and available from peers.

csrftokens.txt A list of recently issued CSRF tokens (for protection against browser cross site request forgery).

1.7.3 Config File Format

The following shows the default configuration file:

```
<configuration version="2">
  <folder id="default" directory="/Users/jb/Sync" ro="false" ignorePerms="false">
    <device id="GXN5ECCWTA2B7EB5FXYL5OWGOADX5EF5VNJAQSIBAY6XHJ24BNOA"></device>
  </folder>
  <device id="GXN5ECCWTA2B7EB5FXYL5OWGOADX5EF5VNJAQSIBAY6XHJ24BNOA" name="jborg-mbp">
    <address>dynamic</address>
  </device>
  <gui enabled="true" tls="true">
    <address>127.0.0.1:54096</address>
    <user>jb</user>
    <password>$2a$10$EKaTicpz2...</password>
    <apikey>080CDOJ9LVUVCMHFK20JD04T882735</apikey>
  </gui>
  <options>
    <listenAddress>:54097</listenAddress>
    <globalAnnounceServer>announce.syncthing.net:22025</globalAnnounceServer>
    <globalAnnounceEnabled>true</globalAnnounceEnabled>
    <localAnnounceEnabled>true</localAnnounceEnabled>
    <parallelRequests>16</parallelRequests>
    <maxSendKbps>0</maxSendKbps>
    <rescanIntervals>60</rescanIntervals>
    <reconnectionIntervals>60</reconnectionIntervals>
    <maxChangeKbps>10000</maxChangeKbps>
    <startBrowser>true</startBrowser>
    <upnpEnabled>true</upnpEnabled>
    <urAccepted>0</urAccepted>
  </options>
</configuration>
```

configuration

This is the root element.

version The config version. The current version is 2.

folder

One or more **folder** elements must be present in the file. Each element describes one folder.

Within the **folder** element one or more **device** element should be present. These must contain the **id** attribute and nothing else. Mentioned devices are those that will be sharing the folder in question. Each mentioned device must have a separate device element later in the file. It is customary that the local device ID is included in all repositories. Syncthing will currently add this automatically if it is not present in the configuration file.

id The folder ID, must be unique. (mandatory)

directory The directory where the folder is stored on this device; not sent to other devices. (mandatory)

ro True if the folder is read only (will not be modified by Syncthing) on this device. (optional, defaults to `false`)

ignorePerms True if the folder should [ignore permissions](#).

device

One or more `device` elements must be present in the file. Each element describes a device participating in the cluster. It is customary to include a `device` element for the local device; Syncthing will currently add one if it is not present.

id The device ID. This must be written in canonical form, that is without any spaces or dashes. (mandatory)

name A friendly name for the device. (optional)

address The address section is only valid inside of `device` elements. It contains a single address, on one of the following forms:

- IPv4 addresses, IPv6 addresses within brackets, or DNS names, all optionally followed by a port number.
- `dynamic`: The address will be resolved using discovery.

gui

There must be *exactly one* `gui` element.

enabled `true/false`

tls `true/false`: If `true` then the GUI will use HTTPS.

address One or more address elements must be present, containing an `ip:port` listen address.

username Set to require authentication.

password Contains the bcrypt hash of the real password.

apikey If set, this is the API key that enables usage of the REST interface.

Additionally, there must be *exactly one* `options` element. It contains the following configuration settings as children:

listenAddress `host:port` or `:port` string denoting an address to listen for BEP connections. More than one `listenAddress` may be given. (default: `0.0.0.0:22000`)

globalAnnounceServer `host:port` string denoting where a global announce server may be reached. (default: `announce.syncthing.net:22025`)

globalAnnounceEnabled `true/false` (default: `true`)

localAnnounceEnabled `true/false` (default: `true`)

parallelRequests The maximum number of outstanding block requests to have against any given peer. (default: 16)

maxSendKbps Rate limit

rescanIntervalS The number of seconds to wait between each scan for modification of the local repositories. A value of 0 disables the scanner. (default: 60)

reconnectionIntervalS The number of seconds to wait between each attempt to connect to currently unconnected devices. (default: 60)

maxChangeKbps The maximum rate of change allowed for a single file. When this rate is exceeded, further changes to the file are not announced, until the rate is reduced below the limit. (default: 10000)

startBrowser `true/false` (default: `true`)

upnpEnabled `true/false` (default: `true`)

urAccepted Whether the user as accepted to submit anonymous usage data. The default, 0, mean the user has not made a choice, and Syncthing will ask at some point in the future. -1 means no, 1 means yes.

1.8 Community Contributions

This page lists integrations, addons and packagings of Syncthing created by the community. Like all documentation pages, it's wiki editable so please do edit and add your own.

1.8.1 GUI Wrappers

Cross-platform

- <https://github.com/syncthing/syncthing-gtk>
- <https://github.com/alex2108/syncthing-tray> (Tray icon only)

Android

- <https://github.com/syncthing/syncthing-android>

Ubuntu

- <https://github.com/icaruseffect/syncthing-ubuntu-indicator>

Windows

- <https://github.com/iss0/SyncthingTray>
- <https://github.com/bloones/SyncThingWin> (Windows service helper and tray icon)
- <https://github.com/canton7/SyncTrayzor> (Windows host for Syncthing. Installer, auto-start, built-in browser, tray icon, folder watcher, and more)
- <https://github.com/kreischweide/metrothing> (Windows UI to monitor multiple Syncthing instances through the REST API)

OS X

- <https://github.com/m0ppers/syncthing-bar>

Kindle Touch

- <https://github.com/gutenye/syncthing-kindle>

1.8.2 Packages and Bundlings

ArchLinux

- <https://www.archlinux.org/packages/?name=syncthing>
- <https://www.archlinux.org/packages/?name=syncthing-gtk>
- <https://aur.archlinux.org/packages/syncthing-inotify>
- <https://aur.archlinux.org/packages/syncthing-discosrv>

arkOS

Syncthing is included in arkOS, <https://arkos.io/>.

Debian / Ubuntu

- PPA: <https://launchpad.net/~ytwld/+archive/ubuntu/syncthing>
- Syncthing GTK PPA: <https://launchpad.net/~nilarimogard/+archive/ubuntu/webupd8/>
- <https://forum.syncthing.net/t/lxle-a-respin-of-lubuntu-now-has-syncthing-included-by-default/1392>

Docker

- <https://github.com/firecat53/dockerfiles/tree/master/syncthing> (runs Syncthing and/or builds the binary from source)
- https://github.com/firecat53/dockerfiles/tree/master/syncthing_discovery (Global announce server in a container)
- <https://github.com/joeybaker/docker-syncthing/> A fully baked docker container that allows custom config and will keep your settings and data past docker image restarts.

Fedora

- https://github.com/thunderbirdtr/syncthing_rpm (SRPM, binary packages coming)
- <https://copr.fedoraproject.org/coprs/tune2fs/syncthing/> (RPMs, based on https://github.com/thunderbirdtr/syncthing_rpm)

FreeBSD

- <http://www.freshports.org/net/syncthing/>

OpenSUSE

- <http://software.opensuse.org/package/syncthing>

Synology NAS

- <http://packages.synocommunity.com/> Add the URL to Package Center in DSM. (NOTE: This page is not readable in your web browser. You can browse the files at <https://synocommunity.com/packages>). Numerous CPU architectures are supported. SPK's may be older versions, however you can execute a Syncthing version upgrade via Web-GUI after installation on Synology device. As Syncthing is marked as beta by the SynoCommunity, you need to enable beta-versions in the settings.

QNAP NAS

- <http://forum.qnap.com/viewtopic.php?f=320&t=97035&start=45#p429896> QPKG (Qnap Package) Available for ALL models x86, x86_64, Arm (all including new models)

1.8.3 Integrations

REST API Bindings

- Ruby: <https://github.com/retgoat/syncthing-ruby>
- Python: <https://github.com/blakev/python-syncthing> (<https://pypi.python.org/pypi/syncthing>)

Ports

- Swift: <https://github.com/dapperstout/pulse-swift>
- Java: <https://github.com/dapperstout/pulse-java>
- PHP: <https://github.com/cebe/pulse-php-discover> (Only Discovery so far)

Configuration management

- SaltStack: <https://bitbucket.org/StartledPhoenix/saltstack-syncthing>
- Puppet: <https://github.com/whefter/puppet-syncthing>
- Command line interface: <https://github.com/syncthing/syncthing-cli>

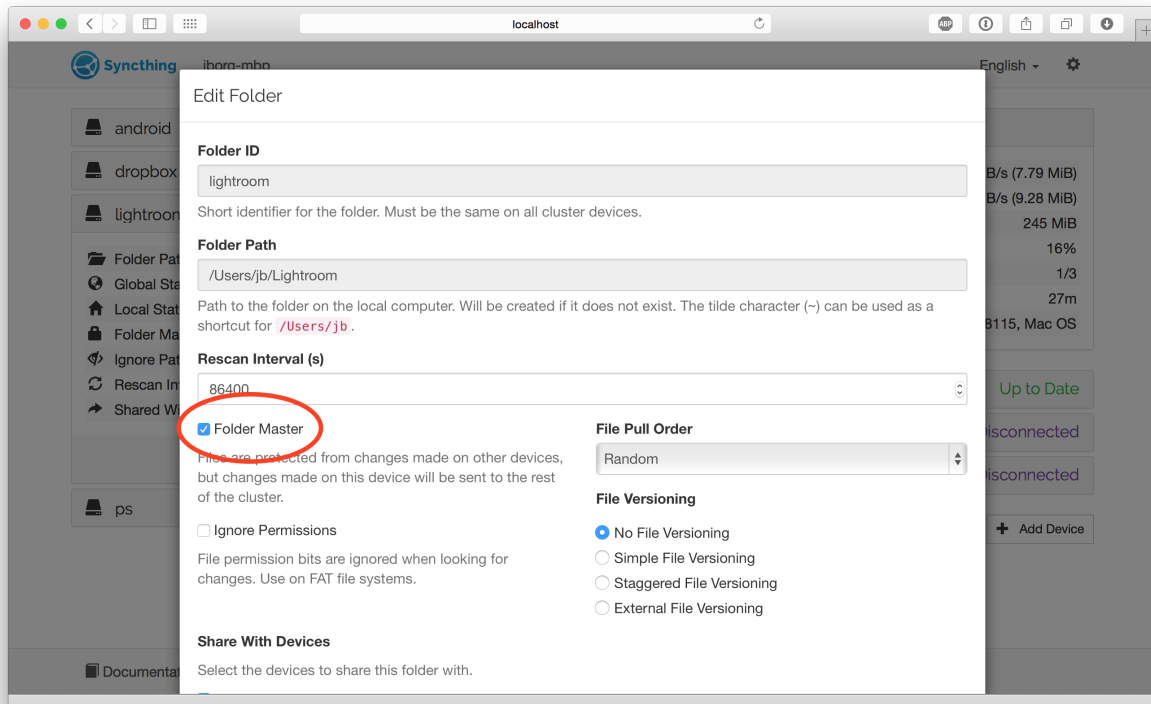
1.9 Folder Master


A folder can be set in “master mode” among the folder settings.








The intention is for this to be used on devices where a “master copy” of files are kept - where the files are not expected to be changed on other devices or where such changes would be undesirable.




In master mode, all changes from other devices in the cluster are ignored. Changes are still *received* so the folder may become “out of sync”, but no changes will be applied.

When a master folder becomes out of sync, a red “Override Changes” button is shown at the bottom of the folder details.



 default Up to Date

 Folder Path	s2
 Global State	6573 items, ~1.86 MiB
 Local State	0 items, ~2.65 MiB
 Out Of Sync	6573 items, ~6.75 KiB
 Folder Master	Yes
 Rescan Interval	15 s
 Shared With	s1, s3

 Override Changes  Rescan  Edit

Clicking this button will make enforce the master's current state on the rest of the cluster. Any changes made to files will be overwritten by the version on the master, any files that don't exist on the master will be deleted, and so on.

1.10 Ignoring Files

1.10.1 Synopsis

`.stignore`

1.10.2 Description

If some files should not be synchronized to other nodes, a file called `.stignore` can be created containing file patterns to ignore. The `.stignore` file must be placed in the root of the repository. The `.stignore` file itself will never be synced to other nodes, although it can `#include` files that *are* synchronized between nodes. All patterns are relative to the repository root.

1.10.3 Patterns

The `.stignore` file contains a list of file or path patterns. The *first* pattern that matches will decide the fate of a given file.

- Regular file names match themselves, i.e. the pattern `foo` matches the files `foo`, `subdir/foo` as well as any directory named `foo`. Spaces are treated as regular characters.
- Asterisk matches zero or more characters in a filename, but does not match the directory separator. `te*st` matches `test`, `subdir/telerest` but not `tele/rest`.
- Double asterisk matches as above, but also directory separators. `te**st` matches `test`, `subdir/telerest` and `tele/sub/dir/rest`.
- Question mark matches a single character that is not the directory separator. `te??st` matches `tebest` but not `teb/st` or `test`.
- A pattern beginning with `/` matches in the current directory only. `/foo` matches `foo` but not `subdir/foo`.
- A pattern beginning with `#include` results in loading patterns from the named file. It is an error for a file to not exist or be included more than once. Note that while this can be used to include patterns from a file in a subdirectory, the patterns themselves are still relative to the repository *root*. Example: `#include more-patterns.txt`.
- A pattern beginning with `!` negates the pattern: matching files are *included* (that is, *not* ignored). This can be used to override more general patterns that follow. Note that files in ignored directories can not be re-included this way. This is due to the fact that syncthing stops scanning when it reaches an ignored directory, so doesn't know what files it might contain.
- A pattern beginning with `(?i)` enables case-insensitive pattern matching. `(?i)test` matches `test`, `TEST` and `tEsT`. The `(?i)` prefix can be combined with other patterns, for example the pattern `(?i)!picture*.png` indicates that `Picture1.PNG` should be synchronized. Note that case-insensitive patterns must start with `(?i)` when combined with other flags.
- A line beginning with `//` is a comment and has no effect.

1.10.4 Example

Given a directory layout:

```
foo
foofoo
bar/
  baz
  quux
  quuz
bar2/
  baz
  frobble
My Pictures/
  Img15.PNG
```

and an .stignore file with the contents:

```
!frobble
!quuz
foo
*2
qu*
(?i)my pictures
```

all files and directories called “foo”, ending in a “2” or starting with “qu” will be ignored. The end result becomes:

```
foo           # ignored, matches "foo"
foofoo        # synced, does not match "foo" but would match "foo*" or "**foo"
bar/          # synced
  baz         # synced
  quux        # ignored, matches "qu*"
  quuz        # synced, matches "qu*" but is excluded by the preceding "!quuz"
bar2/         # ignored, matched "*2"
  baz         # ignored, due to parent being ignored
  frobble     # ignored, due to parent being ignored; "!frobble" doesn't help
My Pictures/  # ignored, matched case insensitive "(?i)my pictures" pattern
  Img15.PNG   # ignored, due to parent being ignored
```

Note: Please note that directory patterns ending with a slash `some/directory/` matches the content of the directory, but not the directory itself. If you want the pattern to match the director and it’s content, make sure it does not have a `/` at the end of the pattern.

1.10.5 Effects on “In Sync” Status

Currently the effects on who is in sync with what can be a bit confusing when using ignore patterns. This should be cleared up in a future version...

Assume two nodes, Alice and Bob, where Alice has 100 files to share, but Bob ignores 25 of these. From Alice’s point of view Bob will become about 75% in sync (the actual number depends on the sizes of the individual files) and remain in “Syncing” state even though it is in fact not syncing anything ([issue #623](#)). From Bob’s point of view it’s 100% up to date but will show fewer files in both the local and global view.

If Bob adds files that have already been synced to the ignore list, they will remain in the “global” view but disappear from the “local” view. The end result is more files in the global repository than in the local, but still 100% in sync ([issue #624](#)). From Alice’s point of view, Bob will remain 100% in sync until the next reconnect, because Bob has already announce that he has the files that are now suddenly ignored.

1.11 Security Principles

Security is one of the primary project goals. This means that it should not be possible for an attacker to join a cluster uninvited, and it should not be possible to extract private information from intercepted traffic. Currently this is implemented as follows.

All traffic is protected by TLS. To prevent uninvited nodes from joining a cluster, the certificate fingerprint of each node is compared to a preset list of acceptable nodes at connection establishment. The fingerprint is computed as the SHA-256 hash of the certificate and displayed in BASE32 encoding to form a reasonably compact and convenient string.

Incoming requests for file data are verified to the extent that the requested file name must exist in the local index and the global model.

For information about ensuring you are running the code you think you are and for reporting security vulnerabilities, please see the official [security page](#).

1.11.1 Information Leakage

Global Discovery

When global discovery is enabled, Syncthing sends an announcement packet every 30 minutes to the global discovery server, so that it can keep a mapping between your device ID and external IP. Also, when connecting to other devices that have not been seen on the local network, a query is sent to the global discovery server containing the device ID of the requested device. The discovery server is currently **hosted by @calmh**. Global discovery defaults to **on**.

When turned off, devices with dynamic addresses not on the local network cannot be found and connected to.

If a different global discovery server is configured, no data is sent to the default global discovery server.

Local Discovery

When local discovery is enabled, Syncthing sends broadcast (IPv4) and multicast (IPv6) packets to the local network every 30 seconds. The packets contain the device ID and listening port. Local discovery defaults to **on**.

An eavesdropper on the local network can deduce which machines are running Syncthing with local discovery enabled, and what their device IDs are.

When turned off, devices with dynamic addresses on the local network cannot be found and connected to.

Upgrade Checks

When automatic upgrades are enabled, Syncthing checks for a new version at startup and then once every twelve hours. This is by an HTTPS request to the download site for releases, currently **hosted at GitHub**. Automatic upgrades default to **on** (unless Syncthing was compiled with upgrades disabled).

Even when automatic upgrades are disabled in the configuration, an upgrade check as above is done when the GUI is loaded, in order to show the “Upgrade to ...” button when necessary. This can be disabled only by compiling syncthing with upgrades disabled.

In effect this exposes the majority of the Syncthing population to tracking by the operator of the download site (currently GitHub). That data is not available to outside parties (including @calmh etc), except that download counts per release binary are available in the GitHub API. The upgrade check (or download) requests *do not* contain any identifiable information about the user, device, Syncthing version, etc.

Usage Reporting

When usage reporting is enabled, Syncthing reports usage data at startup and then every 24 hours. The report is sent as an HTTPS POST to the usage reporting server, currently **hosted by @calmh**. The contents of the usage report can be seen behind the “Preview” link in settings. Usage reporting defaults to **off** but the GUI will ask once about enabling it, shortly after the first install.

The reported data is protected from eavesdroppers, but the connection to the usage reporting server itself may expose the client as running Syncthing.

Sync Connections (BEP)

Sync connections are attempted to all configured devices, when the address is possible to resolve. The sync connection is based on TLS 1.2. The TLS certificates are sent in clear text (as in HTTPS etc), meaning that the certificate Common Name (by default syncthing) is visible.

An eavesdropper can deduce that this is a Syncthing connection and calculate the device ID:s involved based on the hashes of the sent certificates.

Likewise, if the sync port (default 22000) is accessible from the internet, a port scanner may discover it, attempt a TLS negotiation and thus obtain the device certificate. This provides the same information as in the eavesdropper case.

Web GUI

If the web GUI is accessible, it exposes the device as running Syncthing. The web GUI defaults to being reachable from the **local host only**.

1.11.2 In Short

Parties doing surveillance on your network (whether that be corporate IT, the NSA or someone else) will be able to see that you use Syncthing, and your device ID's **are OK to share anyway**, but the actual transmitted data is protected as well as we can. Knowing your device ID can expose your IP address, using global discovery.

1.12 Syncthing

1.12.1 Synopsis

```
syncthing [-audit] [-generate=<dir>] [-gui-address=<address>]
          [-gui-apikey=<key>] [-gui-authentication=<username:password>]
          [-home=<dir>] [-logflags=<flags>] [-no-browser] [-no-restart]
          [-reset] [-upgrade] [-upgrade-check] [-upgrade-to=<url>]
          [-verbose] [-version]
```

1.12.2 Description

Syncthing is an application that lets you synchronize your files across multiple devices. This means the creation, modification or deletion of files on one machine will automatically be replicated to your other devices. We believe your data is your data alone and you deserve to choose where it is stored. Therefore Syncthing does not upload your data to the cloud but exchanges your data across your machines as soon as they are online at the same time.

1.12.3 Options

-audit Write events to audit file.

-generate=<dir> Generate key and config in specified dir, then exit.

-gui-address=<address> Override GUI address.

-gui-apikey=<key> Override GUI API key.

-gui-authentication=<username:password> Override GUI authentication; username:password.

-home=<dir> Set configuration directory. The default configuration directory is: `$HOME/.config/syncthing`.

-logflags=<flags> Select information in log line prefix, default 2. The `-logflags` value is a sum of the following:

- 1: Date
- 2: Time
- 4: Microsecond time
- 8: Long filename
- 16: Short filename

To prefix each log line with date and time, set `-logflags=3` (1 + 2 from above). The value 0 is used to disable all of the above. The default is to show time only (2).

- no-browser** Do not start a browser.
- no-restart** Do not restart; just exit.
- reset** Reset the database.
- upgrade** Perform upgrade.
- upgrade-check** Check for available upgrade.
- upgrade-to=<url>** Force upgrade directly from specified URL.
- verbose** Print verbose log output.
- version** Show version.

1.12.4 Exit Codes

0 Success / Shutdown

1 Error

2 Upgrade not available

3 Restarting

5 Upgrading

Some of these exit codes are only returned when running without a monitor process (with environment variable `STNORESTART` set). Exit codes over 125 are usually returned by the shell/binary loader/default signal handler. Exit codes over 128+N on Unix usually represent the signal which caused the process to exit. For example, 128 + 9 (SIGKILL) = 137.

1.12.5 Development Settings

The following environment variables modify Syncthing's behavior in ways that are mostly useful for developers. Use with care.

STGUIASSETS Directory to load GUI assets from. Overrides compiled in assets.

STTRACE A comma separated string of facilities to trace. The valid facility strings are:

- **beacon**: the beacon package
- **discover**: the discover package
- **events**: the events package
- **files**: the files package
- **http**: the main package; HTTP requests
- **locks**: the sync package; trace long held locks
- **net**: the main package; connections & network messages
- **model**: the model package
- **scanner**: the scanner package
- **stats**: the stats package

- `upnp`: the upnp package
- `xdr`: the xdr package
- `all`: all of the above

STPROFILER Set to a listen address such as “127.0.0.1:9090” to start the profiler with HTTP access.

STCPUPROFILE Write a CPU profile to `cpu-$pid.pprof` on exit.

STHEAPPROFILE Write heap profiles to `heap-$pid-$timestamp.pprof` each time heap usage increases.

STBLOCKPROFILE Write block profiles to `block-$pid-$timestamp.pprof` every 20 seconds.

STPERFSTATS Write running performance statistics to `perf-$pid.csv`. Not supported on Windows.

STNOUPGRADE Disable automatic upgrades.

GOMAXPROCS Set the maximum number of CPU cores to use. Defaults to all available CPU cores.

GOGC Percentage of heap growth at which to trigger GC. Default is 100. Lower numbers keep peak memory usage down, at the price of CPU usage (ie. performance).

1.12.6 See also

[syncthing-config\(5\)](#), [syncthing-stignore\(5\)](#), [syncthing-device-ids\(7\)](#), [syncthing-security\(7\)](#), [syncthing-networking\(7\)](#), [syncthing-versioning\(7\)](#), [syncthing-faq\(7\)](#)

1.13 File Versioning

Warning: This page may be out of date and requires review.

Todo: External versioning requires documenting.

1.13.1 Description

There are 3 types of File Versioning. When you select each in the web interface, a short description of each is shown to help you decide.

Todo: More detail needed here: Can this be a relative path, or must it be an absolute path?

With “Staggered File Versioning” method (only), you would like to specify where removed and deleted files are stored as part of the Versioning feature, you can specify the path in the “Versions Path” input after this method is selected.

1.13.2 No File Versioning

This is the default setting. With no file versioning, files that are replaced or deleted on one device are deleted on other devices that the directory is shared with. (Note: If a folder is marked “Master Folder” on a device, that device will not accept changes to the files in the folder, and therefore will not have files replaced or deleted.)

1.13.3 Simple File Versioning

With “Simple File Versioning” files are moved to the “.stversions” folder (inside your shared folder) when replaced or deleted on a remote device. This option also takes a value in an input titled “Keep Versions” which tells Syncthing how many old versions of the file it should keep. For example, if you set this value to 5, if a file is replaced 5 times on a remote device, you will see 5 time-stamped versions on that file in the “.stversions” folder on the other devices sharing the same folder. With “Simple File Versioning” files are moved to the “.stversions” folder (inside your shared folder) when replaced or deleted on a remote device. This option also takes a value in an input titled “Keep Versions” which tells Syncthing how many old versions of the file it should keep. For example, if you set this value to 5, if a file is replaced 5 times on a remote device, you will see 5 time-stamped versions on that file in the “.stversions” folder on the other devices sharing the same folder.

1.13.4 Staggered File Versioning

With “Staggered File Versioning” files are also moved to the “.stversions” folder (inside your shared folder) when replaced or deleted on a remote device (just like “Simple File Versioning”), however, Version are automatically deleted if they are older than the maximum age or exceed the number of files allowed in an interval.

The following intervals are used and they each have a maximum number of files that will be kept for each.

1 Hour For the first hour, the most recent version is kept every 30 seconds.

1 Day For the first day, the most recent version is kept every hour.

30 Days For the first 30 days, the most recent version is kept every day.

Until Maximum Age The maximum time to keep a version in days. For example, to keep replaced or deleted files in the “.stversions” folder for an entire year, use 365. If only 10 days, use 10. **Note: Set to 0 to keep versions forever.**

1.14 Building Syncthing

Note: You probably only need to go through the build process if you are going to do development on Syncthing or if you need to do a special packaging of it. For all other purposes we recommend using the the official binary releases instead.

Note: If you’re on Linux and want the quickest possible start, check out *Building with Docker*. Otherwise follow the guide below to set up the development environment on your computer.

1.14.1 Branches and Tags

You should base your work on the `master` branch when doing your development. This branch is usually what will be going into the next release and always what pull requests should be based on.

If you're looking to build and package a release of Syncthing you should instead use the latest tag (`vX.Y.Z`) as the contents of `master` may be unstable and unsuitable for general consumption.

1.14.2 Prerequisites

- Go **1.3** or higher
- Git

If you're not a Go developer since before, the easiest way to get going is to download the latest version of Go as instructed in <http://golang.org/doc/install> and `export GOPATH=~`.

Note: You need to set `GOPATH` correctly and the source **must** be checked out into `$GOPATH/src/github.com/syncthing/syncthing`. The instructions below accomplish this correctly.

1.14.3 Building (Unix)

- Install the prerequisites.
- Open a terminal.

```
# This should output "go version go1.3" or higher.
$ go version

# Go is particular about file locations; use this path unless you know very
# well what you're doing.
$ mkdir -p ~/src/github.com/syncthing
$ cd ~/src/github.com/syncthing
# Note that if you are building from a source code archive, you need to
# rename the directory from syncthing-XX.YY.ZZ to syncthing
$ git clone https://github.com/syncthing/syncthing

# Now we have the source. Time to build!
$ cd syncthing

# You should be inside ~/src/github.com/syncthing/syncthing right now.
$ go run build.go
```

Unless something goes wrong, you will have a `syncthing` binary built and ready in `~/src/github.com/syncthing/syncthing/bin`.

1.14.4 Building (Windows)

- Install the prerequisites.
- Open a cmd Window:

```
# This should output "go version go1.3" or higher.
> go version

# Go is particular about file locations; use this path unless you know very
# well what you're doing.
> mkdir c:\src\github.com\syncthing
> cd c:\src\github.com\syncthing
# Note that if you are building from a source code archive, you need to
# rename the directory from syncthing-XX.YY.ZZ to syncthing
> git clone https://github.com/syncthing/syncthing

# Now we have the source. Time to build!
> cd syncthing
> go run build.go
```

Unless something goes wrong, you will have a `syncthing.exe` binary built and ready in `c:\src\github.com\syncthing\syncthing\bin`.

1.14.5 Subcommands and Options

The following `build.go` subcommands and options exist.

- `go run build.go install` – installs binaries in `./bin` (default command, this is what happens when `build.go` is run without any commands or parameters).
- `go run build.go build` – forces a rebuild of the binary to the current directory; similar to `install` but slower.
- `go run build.go clean` – remove build artefacts, guaranteeing a complete rebuild. Use this when switching between normal builds and noupgrade builds.
- `go run build.go test` – run the tests.
- `go run build.go tar` – create a Syncthing `tar.gz` dist file in the current directory. Assumes a Unixy build.
- `go run build.go zip` – create a Syncthing `zip` dist file in the current directory. Assumes a Windows build.
- `go run build.go assets` – rebuild the compiled-in GUI assets.
- `go run build.go deps` – update the in-repo dependencies.
- `go run build.go xdr` – regenerate the XDR en/decoders. Only necessary when the protocol has changed.

The options `-no-upgrade`, `-goos` and `-goarch` can be given to influence `install`, `build`, `tar` and `zip`. Examples:

- `go run build.go -goos linux -goarch 386 tar` – build a `tar.gz` distribution of Syncthing for linux-386.
- `go run build.go -goos windows -no-upgrade zip` – build a `zip` distribution of Syncthing for Windows (current architecture) with upgrading disabled.

Note: Building for a different operating system or architecture than your native one requires Go having been set up for cross compilation. The easiest way to get this right is to use the official Docker image, described below.

1.14.6 Building without Git

Syncthing can be built perfectly fine from a source tarball of course. If the tarball is from our build server it contains a file called `RELEASE` that informs the build system of the version being built. If you're building from a different source package, for example one automatically generated by Github, you must instead pass the `-version` flag to `build.go`.

If you are building something that will be installed as a package (Debian, RPM, ...) you almost certainly want to use `-no-upgrade` as well to prevent the built in upgrade system from being activated.

- `go run build.go -version v0.10.26 -no-upgrade tar` – build a tar.gz distribution of syncthing for the current OS/arch, tagged as `v0.10.26`, with upgrades disabled.

1.14.7 Building with Docker

The Docker based build image exactly replicates the official build process and is a quick way to get up and running with the full cross compiled setup. Start by getting the build image. It is fairly large (about 2 GiB).

```
$ docker pull syncthing/build:latest
```

Then check out and build Syncthing.

```
$ git clone https://github.com/syncthing/syncthing
$ cd syncthing
$ ./build.sh docker-all
```

A full build is done for all supported architectures, and tests are run. The process should end with a bunch of release files (`.tar.gz` and `.zip`) created.

1.15 Debugging Syncthing

There's a lot that happens behind the covers, and Syncthing is generally quite silent about it. A number of environment variables can be used to set the logging to verbose for various parts of the program, and to enable profiling.

1.15.1 Environment Variables

STTRACE

The environment variable `STTRACE` can be set to a comma separated list of "facilities", to enable extra debugging information for said facility. A facility generally maps to a Go package, although there are a few extra that map to parts of the main package. Currently, the following facilities are supported (an up to date list is always printed by `syncthing --help`):

- `beacon` (the beacon package)
- `discover` (the discover package)
- `events` (the events package)
- `files` (the files package)
- `http` (the main package; HTTP requests)
- `net` (the main package; connections & network messages)
- `model` (the model package)

- `scanner` (the scanner package)
- `stats` (the stats package)
- `upnp` (the upnp package)
- `xdr` (the xdr package)
- `all` (all of the above)

The debug output is often of the kind that it doesn't make much sense without looking at the code. The purpose of the different packages / facilities are something like this:

- `beacon` sends and receives UDP broadcasts used by the local discovery system. Debugging here will show which interfaces and addresses are selected for broadcasts, etc.
- `discover` sends and received local discovery packets. Debugging here will output the parsed packets, nodes that get registered etc.
- `files` keeps track of lists of files with metadata and figures out which is the newest version of each.
- `net` shows connection attempts, incoming connections, and the low level error when connection attempts fail.
- `model` is the largest chunk of the system; this is where pulling of out of date files happen, indexes sent and received, and incoming requests for file chunks are logged.
- `scanner` is the local filesystem scanner. Debugging here will output information about changed and unchanged files.
- `upnp` is the upnp talker.
- `xdr` is the low level protocol encoder. Debugging here will output all bytes sent/received over the sync connection. Very verbose.
- `all` simply enabled debugging of all facilities.

Enabling any of the facilities will also change the log format to include microsecond timestamps and file names plus line numbers. This can be used to enable this extra information on the normal logging level, without enabling any debugging: `STTRACE=somethingnonexistent` for example.

Under Unix (including Mac) the easiest way to run Syncthing with an environment variable set is to prepend the variable to the command line. I.e:

```
$ STTRACE=model syncthing
```

On windows, it needs to be set prior to running Syncthing.

```
C:\> set STTRACE=model  
C:\> syncthing
```

STPROFILER

The `STPROFILER` environment variable sets the listen address for the HTTP profiler. If set to for example `:9090` the profiler will start and listen on port 9090. <http://localhost:9090/debug/pprof> is then the address to the profiler. See go tool `pprof` for more information.

STGUIASSETS

Directory to load GUI assets from. Overrides compiled in assets. Useful for developing webgui, commonly use `STGUIASSETS=gui bin/syncthing`

STCPUPROFILE

Write a CPU profile to `cpu-$pid.pprof` on exit.

STHEAPPROFILE

Write heap profiles to `heap-$pid-$timestamp.pprof` each time heap usage increases.

STBLOCKPROFILE

Write block profiles to `block-$pid-$timestamp.pprof` every 20 seconds.

STPERFSTATS

Write running performance statistics to `perf-$pid.csv`. Not supported on Windows.

STNOUPGRADE

Disable automatic upgrades.

GOMAXPROCS

Set the maximum number of CPU cores to use. Defaults to all available CPU cores.

GOGC

Percentage of heap growth at which to trigger GC. Default is 100. Lower numbers keep peak memory usage down, at the price of CPU usage (ie. performance)

1.16 Understanding Device IDs

1.16.1 Description

Every device is identified by a device ID. The device ID is used for address resolution, authentication and authorization. The term “device ID” could interchangeably have been “key ID” since the device ID is a direct properties of the public key in use.

1.16.2 Keys

To understand device IDs we need to look at the underlying mechanisms. At first startup, Syncthing will create an public/private key pair.

Currently this is a 3072 bit RSA key. The keys are saved in the form of the private key (`key.pem`) and a self signed certificate (`cert.pem`). The self signing part doesn't actually add any security or functionality as far as Syncthing is concerned but it enables the use of the keys in a standard TLS exchange.

The typical certificate will look something like this, inspected with `openssl x509`:

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 0 (0x0)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: CN=syncthing
    Validity
      Not Before: Mar 30 21:10:52 2014 GMT
      Not After : Dec 31 23:59:59 2049 GMT
    Subject: CN=syncthing
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (3072 bit)
        Modulus (3072 bit):
          00:da:83:8a:c0:95:af:0a:42:af:43:74:65:29:f2:
          30:e3:b9:12:d2:6b:70:93:da:0b:7b:8a:1e:e5:79:
          ...
          99:09:4c:a9:7b:ba:4a:6a:8b:3b:e6:e7:c7:2c:00:
          90:aa:bc:ad:94:e7:80:95:d2:1b
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
      X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
      X509v3 Basic Constraints: critical
        CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
      68:72:43:8b:83:61:09:68:f0:ef:f0:43:b7:30:a6:73:1e:a8:
      d9:24:6c:2d:b4:bc:c9:e8:3e:0b:1e:3c:cc:7a:b2:c8:f1:1d:
      ...
      88:7e:e2:61:aa:4c:02:e3:64:b0:da:70:3a:cd:1c:3d:86:db:
      df:54:b9:4e:be:1b
```

We can see here that the certificate is little more than a container for the public key; the serial number is zero and the Issuer and Subject are both “syncthing” where a qualified name might otherwise be expected.

An advanced user could replace the `key.pem` and `cert.pem` files with a keypair generated directly by the `openssl` utility or other mechanism.

1.16.3 Device IDs

To form a device ID the SHA-256 hash of the certificate data in DER form is calculated. This means the hash covers all information under the `Certificate:` section above.

The hashing results in a 256 bit hash, which we encode using base32. Base32 encodes five bits per character, so we need $256 / 5 = 51.2$ characters to encode the device ID. This becomes 52 characters in practice, but 52 characters of base32 would decode to 260 bits which is not a whole number of bytes. The base32 encoding adds padding to 280 bits (the next multiple of both 5 and 8 bits) so the resulting ID looks something like `MFZWI3DBONSGYYLTMRWGC43ENRQXGZDMMFZWI3DBONSGYYLTMRWA====`.

The padding (====) is stripped away, the device ID split in four groups, and `check digits` are added for each group. For presentation purposes the device ID is grouped with dashes, resulting in the final value: `MFZWI3D-BONSGYC-YLTMRWG-C43ENR5 -QXGZDMM-FZWI3DP-BONSGYY-LTMRWAD`.

Connection Establishment

So now we know what device IDs are, here's how they are used in Syncthing. When you add a device ID to the syncthing configuration, Syncthing will attempt to connect to that device. The first thing we need to do is figure out the IP and port to connect to. There's three possibilities here;

- The IP and port can be set statically in the configuration. The IP can equally well be a hostname, so if you have a static IP or a dynamic DNS setup this might be a good option.
- Using local discovery, if enabled. Every Syncthing instance on a LAN periodically broadcasts information about itself (device ID, address, port number). If we've seen one of these broadcasts for a given device ID that's where we try to connect.
- Using global discovery, if enabled. Every Syncthing instance announces itself to the global discovery service (device ID and external port number - the internal address is not announced to the global server). If we don't have a static address and haven't seen any local announcements the global discovery server will be queried for an address.

Once we have an address and port a TCP connection is established and a TLS handshake performed. As part of the handshake both devices present their certificates. Once the handshake has completed and the peer certificate is known, the following steps are performed.

1. Calculate the remote device ID by using the process above on the received certificate.
2. Weed out a few possible misconfigurations - i.e. if the device ID is that of the local device or of a device we already have an active connection to. Drop the connection in these cases.
3. Verify the remote device ID against the configuration. If it is not a device ID we are expecting to talk to, drop the connection.
4. Verify the certificate `CommonName` against the configuration. By default, we expect it to be `syncthing`, but when using custom certificates this can be changed.
5. If everything checks out so far, accept the connection.

1.16.4 An Aside About Collisions

The SHA-256 hash is cryptographically collision resistant. This means that there is no way that we know of to create two different messages with the same hash.

You can argue that of course there are collisions - there's an infinite amount of inputs and a finite amount of outputs, so per definition there are infinitely many messages that result in the same hash.

I'm going to quote [stack overflow](#) here:

The usual answer goes thus: what is the probability that a rogue asteroid crashes on Earth within the next second, obliterating civilization-as-we-know-it, and killing off a few billion people? It can be argued that any unlucky event with a probability lower than that is not actually very important.

If we have a "perfect" hash function with output size n , and we have p messages to hash (individual message length is not important), then probability of collision is about $p^2/2n+1$ (this is an approximation which is valid for "small" p , i.e. substantially smaller than $2n/2$). For instance, with SHA-256 ($n=256$) and one billion messages ($p=10^9$) then the probability is about $4.3 \cdot 10^{-60}$.

A mass-murderer space rock happens about once every 30 million years on average. This leads to a probability of such an event occurring in the next second to about 10^{-15} . That's 45 orders of magnitude more probable than the SHA-256 collision. Briefly stated, if you find SHA-256 collisions scary then your priorities are wrong.

It's also worth noting that the property of SHA-256 that we are using is not simply collision resistance but resistance to a preimage attack. I.e. even if you can find two messages that result in a hash collision that doesn't help you attack Syncthing (or TLS in general). You need to create a message that hashes to exactly the hash that my certificate already has or you won't get in.

Note also that it's not good enough to find a random blob of bits that happen to have the same hash as my certificate. You need to create a valid DER- encoded, signed certificate that has the same hash as mine. The difficulty of this is staggeringly far beyond the already staggering difficulty of finding a SHA-256 collision.

1.16.5 Problems and Vulnerabilities

As far as I know, these are the issues or potential issues with the above mechanism.

Discovery Spoofing

Currently, neither the local nor global discovery mechanism is protected by crypto. This means that any device can in theory announce itself for any device ID and potentially receive connections for that device.

This could be a denial of service attack (we can't find the real device for a given device ID, so can't connect to it and sync). It could also be an intelligence gathering attack; if I spoof a given ID, I can see which devices try to connect to it.

It could be mitigated in several ways;

- Announcements could be signed by the device private key. This requires already having the public key to verify.
- Announcements to the global announce server could be done using TLS, so the server calculates the device ID based on the certificate instead of trusting to the device to tell the truth.
- The user could statically configure IP or hostname for the devices.
- The user could run a trusted global server.

It's something we might want to look at at some point, but not a huge problem as I see it.

Long Device IDs are Painful

It's a mouthful to read over the phone, annoying to type into an SMS or even into a computer. And it needs to be done twice, once for each side.

This isn't a vulnerability as such, but a user experience problem. There are various possible solutions:

- Use shorter device IDs with verification based on the full ID ("You entered MFZWI3; I found and connected to a device with the ID MFZWI3-DBONSG-YYLTMR-WGC43E-NRQXGZ-DMMFZW-I3DBON-SGY YLT-MRWA, please confirm that this is correct.").
- Use shorter device IDs with an out of band authentication, a la Bluetooth pairing. You enter a one time PIN into Syncthing and give that PIN plus a short device ID to another user. On initial connect, both sides verify that the other knows the correct PIN before accepting the connection.

1.17 Event API

1.17.1 Description

Syncthing provides a simple long polling interface for exposing events from the core utility towards a GUI.

To receive events, perform a HTTP GET of `/rest/events?since=<lastSeenID>`, where `<lastSeenID>` is the ID of the last event you've already seen or zero. Syncthing returns a JSON encoded array of event objects, starting at the event just after the one with the last seen ID. There is a limit to the number of events buffered, so if the rate of events is high or the time between polling calls is long some events might be missed. This can be detected by noting a discontinuity in the event IDs.

If no new events are produced since `<lastSeenID>`, the HTTP call blocks and waits for new events to happen before returning, or if no new events are produced within 60 seconds, times out.

To receive only a limited number of events, add the `limit=n` parameter with a suitable value for `n` and only the *last* `n` events will be returned. This can be used to catch up with the latest event ID after a disconnection for example: `/rest/events?since=0&limit=1`.

1.17.2 Event Structure

Each event is represented by an object similar to the following:

```
{
  "id": 2,
  "type": "DeviceConnected",
  "time": "2014-07-13T21:04:33.687836696+02:00",
  "data": {
    "addr": "172.16.32.25:22000",
    "id": "NFGKEKE-7Z6RTH7-I3PRZXS-DEJF3UJ-FRWJBFO-VBBDND-4SGNGVZ-QUQHJAG"
  }
}
```

The top level keys `id`, `time`, `type` and `data` are always present, though `data` may be `null`.

id A monotonically increasing integer. The first event generated has id 1, the next has id 2 etc.

time The time the event was generated.

type Indicates the type of (i.e. reason for) the event and is one of the event types below.

data An object containing optional extra information; the exact structure is determined by the event type.

1.17.3 Events

ConfigSaved

Emitted after the config has been saved by the user or by Syncthing itself.

```
{
  "id": 50,
  "type": "ConfigSaved",
  "time": "2014-12-13T00:09:13.5166486Z",
  "data": {
    "Version": 7,
    "Options": { ... },
    "GUI": { ... },
    "Devices": [ ... ],
    "Folders": [ ... ]
  }
}
```

DeviceConnected

Generated each time a connection to a device has been established.

```
{
  "id": 2,
  "type": "DeviceConnected",
  "time": "2014-07-13T21:04:33.687836696+02:00",
  "data": {
    "addr": "172.16.32.25:22000",
    "id": "NFGKEKE-7Z6RTH7-I3PRZXS-DEJF3UJ-FRWJBFO-VBBDND-4SGNGVZ-QUQHJAG"
  }
}
```

DeviceDisconnected

Generated each time a connection to a device has been terminated.

```
{
  "id": 48,
  "type": "DeviceDisconnected",
  "time": "2014-07-13T21:18:52.859929215+02:00",
  "data": {
    "error": "unexpected EOF",
    "id": "NFGKEKE-7Z6RTH7-I3PRZXS-DEJF3UJ-FRWJBFO-VBBDND-4SGNGVZ-QUQHJAG"
  }
}
```

Note: The error key contains the cause for disconnection, which might not necessarily be an error as such. Specifically, “EOF” and “unexpected EOF” both signify TCP connection termination, either due to the other device restarting or going offline or due to a network change.

DeviceDiscovered

Emitted when a new device is discovered using local discovery.

```
{
  "id": 13,
  "type": "DeviceDiscovered",
  "time": "2014-07-17T13:28:05.043465207+02:00",
  "data": {
    "addrs": [
      "172.16.32.25:22000"
    ],
    "device": "NFGKEKE-7Z6RTH7-I3PRZXS-DEJF3UJ-FRWJBFO-VBBTDND-4SGNGVZ-QUQHJAG"
  }
}
```

DeviceRejected

Emitted when there is a connection from a device we are not configured to talk to.

```
{
  "id": 24,
  "type": "DeviceRejected",
  "time": "2014-08-19T10:43:00.562821045+02:00",
  "data": {
    "address": "127.0.0.1:51807",
    "device": "EJHMPAQ-OGCVORE-ISB4IS3-SYYVJXF-TKJGLTU-66DIQPF-GJ5D2GX-GQ3OWQK"
  }
}
```

DownloadProgress

Emitted during file downloads for each folder for each file. By default only a single file in a folder is handled at the same time, but custom configuration can cause multiple files to be shown.

```
{
  "id": 221,
  "type": "DownloadProgress",
  "time": "2014-12-13T00:26:12.9876937Z",
  "data": {
    "folder1": {
      "file1": {
        "Total": 800,
        "Pulling": 2,
        "CopiedFromOrigin": 0,

```

(continues on next page)

(continued from previous page)

```

        "Reused": 633,
        "CopiedFromElsewhere": 0,
        "Pulled": 38,
        "BytesTotal": 104792064,
        "BytesDone": 87883776
    },
    "dir\\file2": {
        "Total": 80,
        "Pulling": 2,
        "CopiedFromOrigin": 0,
        "Reused": 0,
        "CopiedFromElsewhere": 0,
        "Pulled": 32,
        "BytesTotal": 10420224,
        "BytesDone": 4128768
    }
},
"folder2": {
    "file3": {
        "Total": 800,
        "Pulling": 2,
        "CopiedFromOrigin": 0,
        "Reused": 633,
        "CopiedFromElsewhere": 0,
        "Pulled": 38,
        "BytesTotal": 104792064,
        "BytesDone": 87883776
    },
    "dir\\file4": {
        "Total": 80,
        "Pulling": 2,
        "CopiedFromOrigin": 0,
        "Reused": 0,
        "CopiedFromElsewhere": 0,
        "Pulled": 32,
        "BytesTotal": 10420224,
        "BytesDone": 4128768
    }
}
}
}

```

- Total - total number of blocks in the file
- Pulling - number of blocks currently being downloaded
- CopiedFromOrigin - number of blocks copied from the file we are about to replace
- Reused - number of blocks reused from a previous temporary file
- CopiedFromElsewhere - number of blocks copied from other files or potentially other folders
- Pulled - number of blocks actually downloaded so far
- BytesTotal - approximate total file size

- BytesDone - approximate number of bytes already handled (already reused, copied or pulled)

Where block size is 128KB.

Files/folders appearing in the event data imply that the download has been started for that file/folder, where disappearing implies that the downloads has been finished or failed for that file/folder. There is always a last event emitted with no data, which implies all downloads being finished/failed.

FolderCompletion

The FolderCompletion event is emitted when the local or remote contents for a folder changes. It contains the completion percentage for a given remote device and is emitted once per currently connected remote device.

```
{
  "id": 84,
  "type": "FolderCompletion",
  "time": "2015-04-17T14:14:27.043576583+09:00",
  "data": {
    "completion": 100,
    "device": "I6KAH76-66SLLB-5PFXSOA-UFJCDZC-YAOMLEK-CP2GB32-BV5RQST-3PSROAU",
    "folder": "default"
  }
}
```

FolderRejected

Emitted when a device sends index information for a folder we do not have, or have but do not share with the device in question.

```
{
  "id": 27,
  "type": "FolderRejected",
  "time": "2014-08-19T10:41:06.761751399+02:00",
  "data": {
    "device": "EJHMPAQ-OGCVORE-ISB4IS3-SYYVJXF-TKJGLTU-66DIQPF-GJ5D2GX-GQ3OWQK",
    "folder": "unique"
  }
}
```

FolderSummary

The FolderSummary event is emitted when folder contents have changed locally. This can be used to calculate the current local completion state.

```
{
  "id": 16,
  "type": "FolderSummary",
  "time": "2015-04-17T14:12:20.460121585+09:00",
  "data": {
    "folder": "default",
    "summary": {
      "globalBytes": 0,

```

(continues on next page)

(continued from previous page)

```
    "globalDeleted": 0,
    "globalFiles": 0,
    "ignorePatterns": false,
    "inSyncBytes": 0,
    "inSyncFiles": 0,
    "invalid": "",
    "localBytes": 0,
    "localDeleted": 0,
    "localFiles": 0,
    "needBytes": 0,
    "needFiles": 0,
    "state": "idle",
    "stateChanged": "2015-04-17T14:12:12.455224687+09:00",
    "version": 0
  }
}
```

ItemFinished

Generated when Syncthing ends synchronizing a file to a newer version. A successful operation:

```
{
  "id": 93,
  "type": "ItemFinished",
  "time": "2014-07-13T21:22:03.414609034+02:00",
  "data": {
    "item": "test.txt",
    "folder": "default",
    "error": null,
    "type": "file",
    "action": "update"
  }
}
```

An unsuccessful operation:

```
{
  "id": 44,
  "type": "ItemFinished",
  "time": "2015-05-27T11:21:05.711133004+02:00",
  "data": {
    "action": "update",
    "error": "open /Users/jb/src/github.com/syncthing/syncthing/test/s2/foo/.
↪syncthing.hej.tmp: permission denied",
    "folder": "default",
    "item": "foo/hej",
    "type": "file"
  }
}
```

The action field is either update or delete.

ItemStarted

Generated when Syncthing begins synchronizing a file to a newer version.

```
{
  "id": 93,
  "type": "ItemStarted",
  "time": "2014-07-13T21:22:03.414609034+02:00",
  "data": {
    "item": "test.txt",
    "folder": "default",
    "type": "file",
    "action": "update"
  }
}
```

The action field is either update or delete.

LocalIndexUpdated

Generated when the local index information has changed, due to synchronizing one or more items from the cluster or discovering local changes during a scan.

```
{
  "id": 59,
  "type": "LocalIndexUpdated",
  "time": "2014-07-17T13:27:28.051369434+02:00",
  "data": {
    "folder": "default",
    "items": 1000,
  }
}
```

Ping

The Ping event is generated automatically every 60 seconds. This means that even in the absence of any other activity, the event polling HTTP request will return within a minute.

```
{
  "id": 46,
  "type": "Ping",
  "time": "2014-07-13T21:13:18.502171586+02:00",
  "data": null
}
```

RemoteIndexUpdated

Generated each time new index information is received from a device.

```
{
  "id": 44,
  "type": "RemoteIndexUpdated",
  "time": "2014-07-13T21:04:35.394184435+02:00",
  "data": {
    "device": "NFGKEKE-7Z6RTH7-I3PRZXS-DEJF3UJ-FRWJBFO-VBBTDND-4SGNGVZ-QUQHJAG",
    "folder": "lightroom",
    "items": 1000
  }
}
```

Starting

Emitted exactly once, when Syncthing starts, before parsing configuration etc.

```
{
  "id": 1,
  "type": "Starting",
  "time": "2014-07-17T13:13:32.044470055+02:00",
  "data": {
    "home": "/home/jb/.config/syncthing"
  }
}
```

StartupCompleted

Emitted exactly once, when initialization is complete and Syncthing is ready to start exchanging data with other devices.

```
{
  "id": 1,
  "type": "StartupComplete",
  "time": "2014-07-13T21:03:18.383239179+02:00",
  "data": null
}
```

StateChanged

Emitted when a folder changes state. Possible states are `idle`, `scanning`, `cleaning` and `syncing`. The field `duration` is the number of seconds the folder spent in state `from`. In the example below, the folder `default` was in state `scanning` for 0.198 seconds and is now in state `idle`.

```
{
  "id": 8,
  "type": "StateChanged",
  "time": "2014-07-17T13:14:28.697493016+02:00",
  "data": {
```

(continues on next page)

(continued from previous page)

```

    "folder": "default",
    "from": "scanning",
    "duration": 0.197828699000000002,
    "to": "idle"
  }
}

```

1.18 Issue Management

Bugs, feature requests and other things we need to do are tracked as Github issues. Issues can be of various types and in various states, and also belong to milestones or not. This page is an attempt to document the current practice.

1.18.1 Labels

Issues without labels are undecided - that is, we don't yet know if it's a bug, a configuration issue, a feature request or what. Issues that are invalid for whatever reason are closed with a short explanation of why. Examples include "Duplicate of #123", "Discovered to be configuration error", "Rendered moot by #123" and so on. We don't use the "invalid" or "wontfix" labels.

- **android** - Marks an issue as occurring on the Android platform only.
- **bug** - The issue is a verified bug.
- **build** - The issue is caused by or requires changes to the build system (scripts or Docker image).
- **docs** - Something requires documenting.
- **easy** - This could be easily fixed, probably an hour's work or less. These issues are good starting points for new contributors.
- **enhancement** - This is a new feature or an improvement of some kind, as opposed to a problem (bug).
- **help-wanted** - The core team can't or won't do this, but someone else is welcome to. This does not mean that help is not wanted on the *other* issues. You can see this as a soft **wontfix**. (A hard **wontfix** is simply a close with a short explanation why.)
- **pr-bugfix** - This pull request *fixes* a bug. This is different from the **bug** label, as there may also be pull requests with for example tests that *prove* a bug which would then be labeled **bug**.
- **pr-refactor** - This pull request is a refactoring, i.e. not supposed to change behavior.
- **pr-wait-or-pending** - This pull request is not ready for merging, even if the tests pass and it looks good. It is incomplete or requires more discussion.
- **protocol** - This requires a change to the protocol.

1.18.2 Milestone

There are milestones for major and sometimes minor versions. An issue being assigned to a milestone means it is a blocker - the release can't be made without the issue being closed. Issues not assigned to a milestone can be handled whenever.

1.18.3 Assignee

Users can be assigned to issues. We don't usually do so. Sometimes someone assigns themselves to an issue to indicate "I'm working on this" to avoid others doing so too. It's not mandatory.

1.18.4 Locking

We don't normally lock issues (prevent further discussion on them). There are some exceptions though;

- "Popular" issues that attract lots of "me too" and "+1" comments. These are noise and annoy people with useless notifications via mail and in the Github interface. Once the issue is clear and it suffers from this symptom I may lock it.
- Contentious bikeshedding discussions. After two sides in a discussion have clarified their points, there is no point arguing endlessly about it. As above, this may get closed.
- Duplicates. Once an issue has been identified as a duplicate of another issue, it may be locked to prevent further discussion there. The intention is to move the discussion to the other (referenced) issue, while someone just doing a search and jumping on the first match might otherwise resurrect discussion in the duplicate.

1.19 Interacting with Jenkins

Jenkins will test pull requests from recognized authors. If the pull request is not from a recognized author, an *admin* needs to tell Jenkins to perform the tests, after giving the patch a manual look over to prevent shenanigans. A number of tests are performed, ranging from verifying correct code formatting and that the author is included in the AUTHORS file to that the code in fact builds and passes tests. A pull request should usually only be merged if all tests return green, although there are exceptions.

To enable testing for this pull request only:

```
@st-jenkins ok to test
```

To enable testing for this pull request, and all future pull requests from the same author:

```
@st-jenkins add to whitelist
```

For pull requests where Jenkins has already run its tests, but should run them again:

```
@st-jenkins test this please
```

This is not necessary when new commits are pushed (tests will be rerun for the new commits), but is useful if something has changed server side or to verify that everything is still OK if *master* has been updated.

1.20 Creating a Release

1.20.1 Prerequisites

- Push access to the `syncthing` repo, for pushing a new tag.
- SSH account on build server, member of the `jenkins` group, for accessing and signing the releases.
- The release signing key on your GPG keyring on your own computer (for signing the tag) and your account on the build server (for signing the release). In a pinch, having it just on the build server will do since you can run `git` there to create, sign and push the tag.
- Your Github token in the `GITHUB_TOKEN` environment variable on the build server, for uploading the release.

1.20.2 Process

Make sure the build seems sane. I.e. the build is clean on the build server, the integration tests pass without complaints. (Currently, the tests are a bit flaky, specifically the `TestSyncCluster...` ones. I'm not sure if the tests are weird or there is something actually bad happening that should be fixed - requires investigation).

Create a new, signed tag on master, with the version as comment, and push it:

```
$ git tag -a -s -u release@syncthing.net -m v0.10.15 v0.10.15
$ git push --tags
```

The build server will build packages under the job `syncthing-release`. Wait for this to complete successfully before moving on.

Run `./changelog.sh` (in the repo) to create the changelog comparison from the previous release. Copy to clipboard.

On the Github releases page, select the newly pushed tag and hit "Edit Tag". Set the "Release title" to the same version as the tag, paste in the changelog from above, and publish the release.

On the build server, logged in via `ssh`, run `/usr/local/bin/upload-release`. This will create the `md5sum` and `sha1sum` files, sign them (`gpg` will prompt for key passphrase twice) and upload the whole shebang to Github.

Verify it looks sane on the releases page.

1.21 Release Schedule

1.21.1 Structure

Syncthing follows the [Semantic Versioning](#) scheme of versioning. Each release has a three part version number: *major.minor.patch*.

A new *major* version is released when there are incompatible API or protocol changes, a new *minor* version is released when there are new features but compatibility with older releases is retained, and a new *patch* version is released when there are bug fixes (compatibility with older releases always retained).

While still in pre-release mode, i.e. versions `0.x`, breaking changes are made in minor releases rather than major. Version `0.7.3` should be able to talk to version `0.7.52`, but will probably not understand version `0.8.0`. This also means that if you don't like `0.7.52`, you can safely downgrade to `0.7.3` again and keep your configuration, index caches, etc. However `0.8.0` might have a different format for those things so a downgrade to `0.7.x` might be trickier.

1.21.2 Patch Releases

A new patch release is made each Sunday, if there have been changes committed since the last release. Serious bugs, such as would crash the client or corrupt data, cause an immediate (out of schedule) patch release.

1.21.3 Minor Releases

Minor releases are made when new functionality is ready for release. This happens approximately once every few weeks, with the pace slowing as the 1.0 release nears.

1.21.4 Major Releases

A new major release is a rare event. At the time of writing this has not yet happened and is foreseen to happen only once in the foreseeable future - the 1.0 release.

1.22 REST API

1.22.1 Description

Syncthing exposes a REST interface over HTTP on the GUI port. This is used by the GUI code (JavaScript) and can be used by other processes wishing to control Syncthing. In most cases both the input and output data is in JSON format. The interface is subject to change.

1.22.2 API Key

To use the POST methods, or *any* method when authentication is enabled, an API key must be set and used. The API key can be generated in the GUI, or set in the `configuration/gui/apikey` element in the configuration file. To use an API key, set the request header `X-API-Key` to the API key value.

1.22.3 System Endpoints

GET /rest/system/config

Returns the current configuration.

```
{  
  # etc  
}
```

GET /rest/system/config/insync

Returns whether the config is in sync, i.e. whether the running configuration is the same as that on disk.

```
{
  "configInSync": true
}
```

POST /rest/system/config

Post the full contents of the configuration, in the same format as returned by the corresponding GET request. The configuration will be saved to disk and the configInSync flag set to false. Restart Syncthing to activate.

GET /rest/system/connections

Returns the list of current connections and some metadata associated with the connection/peer.

```
{
  "connections": {
    "SMAHWLH-AP74FAB-QWLDYGV-Q65ASPL-GAAR2TB-KEF5FLB-DRLZCPN-DJBFZAG": {
      "address": "172.21.20.78:22000",
      "at": "2015-03-16T21:51:38.672758819+01:00",
      "clientVersion": "v0.10.27",
      "inBytesTotal": 415980,
      "outBytesTotal": 396300
    }
  },
  "total": {
    "address": "",
    "at": "2015-03-16T21:51:38.672868814+01:00",
    "clientVersion": "",
    "inBytesTotal": 415980,
    "outBytesTotal": 396300
  }
}
```

GET /rest/system/discovery

Returns the contents of the local discovery cache.

```
{
  "LGFPDIT7SKNNJVJZA4FC7QNCRKCE753K72BW5QD2FOZ7FRFEP57Q": [
    "192.162.129.11:22000"
  ]
}
```

POST /rest/system/discovery/hint

Post with the query parameters `device` and `addr` to add entries to the discovery cache.

```
curl -X POST http://127.0.0.1:8384/rest/system/discovery/hint?
↳device=LGFDPIT7SKNNJVJZA4FC7QNCRKCE753K72BW5QD2FOZ7FRFEP57Q\&addr=192.162.129.11:22000
# Or with the X-API-Key header:
curl -X POST --header "X-API-Key: TcE28kVPdtJ8COws1JdM0b2nodj77WeQ" http://127.0.0.
↳1:8384/rest/system/discovery/hint?
↳device=LGFDPIT7SKNNJVJZA4FC7QNCRKCE753K72BW5QD2FOZ7FRFEP57Q\&addr=192.162.129.11:22000
```

POST /rest/system/error/clear

Post with empty body to remove all recent errors.

GET /rest/system/error

Returns the list of recent errors.

```
{
  "errors": [
    {
      "time": "2014-09-18T12:59:26.549953186+02:00",
      "error": "This is an error string"
    }
  ]
}
```

POST /rest/system/error

Post with an error message in the body (plain text) to register a new error. The new error will be displayed on any active GUI clients.

GET /rest/system/ping

Returns a `{"ping": "pong"}` object.

```
{
  "ping": "pong"
}
```

POST /rest/system/ping

Returns a {"ping": "pong"} object.

Note: Due to being a POST request, this method requires using an API key or CSRF token, as opposed to the GET request to the same URL.

POST /rest/system/reset

Post with empty body to immediately *reset* Syncthing. This means renaming all folder directories to temporary, unique names, wiping all indexes and restarting. This should probably not be used during normal operations...

POST /rest/system/restart

Post with empty body to immediately restart Syncthing.

POST /rest/system/shutdown

Post with empty body to cause Syncthing to exit and not restart.

GET /rest/system/status

Returns information about current system status and resource usage.

```
{
  "alloc": 30618136,
  "cpuPercent": 0.006944836512046966,
  "extAnnounceOK": {
    "udp4://announce.syncthing.net:22026": true,
    "udp6://announce-v6.syncthing.net:22026": true
  },
  "goroutines": 49,
  "myID": "P56IOI7-MZJNU2Y-IQGDREY-DM2MGTI-MGL3BXN-PQ6W5BM-TBBZ4TJ-XZWICQ2",
  "pathSeparator": "/",
  "sys": 42092792,
  "tilde": "/Users/jb"
}
```

GET /rest/system/upgrade

Checks for a possible upgrade and returns an object describing the newest version and upgrade possibility.

```
{
  "latest": "v0.10.27",
  "newer": false,
  "running": "v0.10.27+5-g36c93b7"
}
```

POST /rest/system/upgrade

Perform an upgrade to the newest released version and restart. Does nothing if there is no newer version than currently running.

GET /rest/system/version

Returns the current Syncthing version information.

```
{
  "arch": "amd64",
  "longVersion": "syncthing v0.10.27+3-gea8c3de (go1.4 darwin-amd64 default) jb@syno_
↳ 2015-03-16 11:01:29 UTC",
  "os": "darwin",
  "version": "v0.10.27+3-gea8c3de"
}
```

1.22.4 Database Endpoints

GET /rest/db/browse

Returns the directory tree of the global model. Directories are always JSON objects (map/dictionary), and files are always arrays of modification time and size. The first integer is the files modification time, and the second integer is the file size.

The call takes one mandatory `folder` parameter and two optional parameters. Optional parameter `levels` defines how deep within the tree we want to dwell down (0 based, defaults to unlimited depth) Optional parameter `prefix` defines a prefix within the tree where to start building the structure.

```
$ curl -s http://localhost:8384/rest/db/browse?folder=default | json_pp
{
  "directory": {
    "file": ["2015-04-20T22:20:45+09:00", 130940928],
    "subdirectory": {
      "another file": ["2015-04-20T22:20:45+09:00", 130940928]
    }
  },
  "rootfile": ["2015-04-20T22:20:45+09:00", 130940928]
}

$ curl -s http://localhost:8384/rest/db/browse?folder=default&levels=0 | json_pp
{
  "directory": {},
  "rootfile": ["2015-04-20T22:20:45+09:00", 130940928]
}

$ curl -s http://localhost:8384/rest/db/browse?folder=default&levels=1 | json_pp
{
  "directory": {
    "file": ["2015-04-20T22:20:45+09:00", 130940928],
    "subdirectory": {}
  },
  "rootfile": ["2015-04-20T22:20:45+09:00", 130940928]
}
```

(continues on next page)

(continued from previous page)

```

    "rootfile": ["2015-04-20T22:20:45+09:00", 130940928]
  }

$ curl -s http://localhost:8384/rest/db/browse?folder=default&prefix=directory/
↳ subdirectory | json_pp
{
  "another file": ["2015-04-20T22:20:45+09:00", 130940928]
}

$ curl -s http://localhost:8384/rest/db/browse?folder=default&prefix=directory&levels=0_
↳ | json_pp
{
  "file": ["2015-04-20T22:20:45+09:00", 130940928],
  "subdirectory": {}
}

```

Note: This is an expensive call, increasing CPU and RAM usage on the device. Use sparingly.

GET /rest/db/completion

Returns the completion percentage (0 to 100) for a given device and folder. Takes device and folder parameters.

```

{
  "completion": 0
}

```

Note: This is an expensive call, increasing CPU and RAM usage on the device. Use sparingly.

GET /rest/db/file

Returns most data available about a given file, including version and availability.

```

{
  "availability": [
    "I6KAH76-66SLLB-5PFXSOA-UFJCDZC-YAOMLEK-CP2GB32-BV5RQST-3PSROAU"
  ],
  "global": {
    "flags": "0644",
    "localVersion": 3,
    "modified": "2015-04-20T22:20:45+09:00",
    "name": "util.go",
    "numBlocks": 1,
    "size": 9642,
    "version": [
      "5407294127585413568:1"
    ]
  },
}

```

(continues on next page)

(continued from previous page)

```
"local": {
  "flags": "0644",
  "localVersion": 4,
  "modified": "2015-04-20T22:20:45+09:00",
  "name": "util.go",
  "numBlocks": 1,
  "size": 9642,
  "version": [
    "5407294127585413568:1"
  ]
}
```

GET /rest/db/ignores

Takes one parameter, `folder`, and returns the content of the `.stignore` as the `ignore` field. A second field, `patterns`, provides a compiled version of all included ignore patterns in the form of regular expressions. Excluded items in the `patterns` field have a nonstandard (`?exclude`) marker in front of the regular expression.

```
{
  "ignore": [
    "/Backups"
  ],
  "patterns": [
    "(?i)^Backups$",
    "(?i)^Backups/.*$"
  ]
}
```

POST /rest/db/ignores

Expects a format similar to the output of GET call, but only containing the `ignore` field (`patterns` field should be omitted). It takes one parameter, `folder`, and either updates the content of the `.stignore` echoing it back as a response, or returns an error.

GET /rest/db/need

Takes one parameter, `folder`, and returns lists of files which are needed by this device in order for it to become in sync.

```
{
  # Files currently being downloaded
  "progress": [
    {
      "flags": "0755",
      "localVersion": 6,
      "modified": "2015-04-20T23:06:12+09:00",
      "name": "ls",
      "size": 34640,
```

(continues on next page)

(continued from previous page)

```

    "version": [
        "5157751870738175669:1"
    ]
  },
  # Files queued to be downloaded next (as per array order)
  "queued": [
    ...
  ],
  # Files to be downloaded after all queued files will be downloaded.
  # This happens when we start downloading files, and new files get added while we are
  ↪ downloading.
  "rest": [
    ...
  ]
}

```

POST /rest/db/prio

Moves the file to the top of the download queue.

```
curl -X POST http://127.0.0.1:8384/rest/db/prio?folder=default&file=foo/bar
```

Response contains the same output as GET /rest/db/need

POST /rest/db/scan

Request immediate rescan of a folder, or a specific path within a folder. Takes the mandatory parameter *folder* (folder ID), an optional parameter *sub* (path relative to the folder root) and an optional parameter *next*. If *sub* is omitted or empty, the entire folder is scanned for changes, otherwise only the given path (and children, in case it's a directory) is scanned. The *next* argument delays Syncthing's automated rescan interval for a given amount of seconds.

Requesting scan of a path that no longer exists, but previously did, is valid and will result in Syncthing noticing the deletion of the path in question.

Returns status 200 and no content upon success, or status 500 and a plain text error if an error occurred during scanning.

```
curl -X POST http://127.0.0.1:8384/rest/db/scan?folder=default&sub=foo/bar
```

GET /rest/db/status

Returns information about the current status of a folder.

Parameters: *folder*, the ID of a folder.

```

{
  # latest version according to cluster:
  "globalBytes": 13173473780,
  "globalDeleted": 1847,
  "globalFiles": 42106,
  # what we have locally:

```

(continues on next page)

(continued from previous page)

```
"localBytes": 13173473780,
"localDeleted": 1847,
"localFiles": 42106,
# which part of what we have locally is the latest cluster version:
"inSyncBytes": 13173473780,
"inSyncFiles": 42106,
# which part of what we have locally should be fetched from the cluster:
"needBytes": 0,
"needFiles": 0,
# various other metadata
"ignorePatterns": true,
"invalid": "",
"state": "idle",
"stateChanged": "2015-03-16T21:47:28.750853241+01:00",
"version": 71989
}
```

Note: This is an expensive call, increasing CPU and RAM usage on the device. Use sparingly.

1.22.5 Statistics Endpoints

GET /rest/stats/device

Returns general statistics about devices. Currently, only contains the time the device was last seen.

```
$ curl -s http://localhost:8384/rest/stats/device | json
{
  "P56IOI7-MZJNU2Y-IQGDREY-DM2MGTI-MGL3BXN-PQ6W5BM-TBBZ4TJ-XZWICQ2": {
    "lastSeen" : "2015-04-18T11:21:31.3256277+01:00"
  }
}
```

GET /rest/stats/folder

Returns general statistics about folders. Currently, only contains the last synced file.

```
$ curl -s http://localhost:8384/rest/stats/folder | json
{
  "folderid" : {
    "lastFile" : {
      "filename" : "file/name",
      "at" : "2015-04-16T22:04:18.3066971+01:00"
    }
  }
}
```

1.22.6 Misc Services Endpoints

GET /rest/svc/deviceid

Verifies and formats a device ID. Accepts all currently valid formats (52 or 56 characters with or without separators, upper or lower case, with trivial substitutions). Takes one parameter, `id`, and returns either a valid device ID in modern format, or an error.

```
$ curl -s http://localhost:8384/rest/svc/deviceid?id=1234 | json
{
  "error": "device ID invalid: incorrect length"
}

$ curl -s http://localhost:8384/rest/svc/deviceid?id=p56ioi7m--zjnu2iq-gdr-eydm-
2mgtmgl3bxnpq6w5btbbz4tjxzwicq | json
{
  "id": "P56IOI7-MZJNU2Y-IQGDREY-DM2MGTI-MGL3BXN-PQ6W5BM-TBBZ4TJ-XZWICQ2"
}
```

GET /rest/svc/lang

Returns a list of canonicalized localization codes, as picked up from the `Accept-Language` header sent by the browser.

```
["sv_sv", "sv", "en_us", "en"]
```

GET /rest/svc/report

Returns the data sent in the anonymous usage report.

```
{
  "folderMaxFiles": 42106,
  "folderMaxMiB": 12563,
  "longVersion": "syncthing v0.10.27+5-g36c93b7 (go1.4 darwin-amd64 default) jb@syno_
2015-03-16 20:43:34 UTC",
  "memorySize": 16384,
  "memoryUsageMiB": 41,
  "numDevices": 10,
  "numFolders": 4,
  "platform": "darwin-amd64",
  "sha256Perf": 122.38,
  "totFiles": 45180,
  "totMiB": 18151,
  "uniqueID": "6vulmdGw",
  "version": "v0.10.27+5-g36c93b7"
}
```