# osync v1.2 Documentation

## (C) 2013-2017 by Orsiris de Jong

## 25 March 2017

http://www.netpower.fr/osync

---

Table of Contents

# 1 Introduction

## 1.1 Quickstart guide

osync is a command line two way synchronization tool for Linux / BSD / Android / Busybox / MacOSX and Windows, with an emphasis on reliability and automation on low bandwidth links.

For those who think TL;DR, a quickstart guide can be found in the README.md file.

## 1.2 Basic synchronization problems

Synchronization is usually found in two flavors, bloc level sync and file level sync. While whole bloc level synchronization is generally a good way of doing, it's also very greedy in network ressources and is not easy to setup. That's where file level sync comes in handy, where only some directories & files need to be synced.

Imagine you're syncing two remote offices of a same company. If you're syncing a user's home directory or it's roaming profile as a night task, the next day, the user will find it's roaming profile up to date at the remote office.

But what would happen if two users work on the same file in a public folder, at the same time, on both offices ? Some sync software would stop sync and ask what to do. Others might simply deleted the oldest version of the file, even if it was modified on both sides.

Also, what would happen if a user uploads a lot of data ? If the link between both offices cannot handle enough data transfer in a given time, any other sync task won't be run, and the sync would continue during the day, when bandwidth is necessary elsewhere.

What would happen if a power fault occurs while synchronization is going on ?

## 1.3 What osync can do

### 1.3.1 Making synchronization reliable

osync is designed to synchronize two folders on both local and / or remote systems.

It is time controlled, which means you can decide how much time it should spend on a sync task before stopping it and launching the next one.

It's designed to resume failed or stopped sync tasks in order to gain as much time as possible, or totally restart the sync task if resuming fails.

It can keep multiple versions of a file in case of conflicts.

It handles soft deletion. If a user deletes a file on replica A, it will move that file on replica B to the ".osync_workdir/deleted" folder.

It will automatically clean old files (soft deleted and conflict backups) after a defined amount of days.

It will perform various checks before launching a synchronization, like checking for free disk space against a defined minimum value.

### 1.3.2 Making a sysadmin's life easier

osync is also desgined to ease synchronization setups.

It will trigger an email alert including the whole sync process execution log if a warning / error is found.

Pre-processing and post-processing commands can be launched on local and / or remote systems, which may be useful to launch snapshot software, flush or standby virtual machines, etc).

Multiple concurrent instances of osync can be run as long as they don't sync the same replicas at the same time.

A batch processing script is included (osync-batch.sh) to launch sequential sync tasks. Failed sync tasks are rerun if every other task has completed and there's still some time left in a given timespan.

osync can use rsync or ssh tunnel compression to gain bandwidth. Bandwidth can also be limited for slow link sharing.

It can be run in quicksync mode for the impatient (nothing to configure except the replica paths), or with a full blown config file.

You may run osync manually, schedule it with cron, or have it monitor a directory as a daemon and launch sync tasks on file modifications.

osync has been deeply tested on RHEL / CentOS 5, 6 & 7, Fedora 23, 24 & 25, Debian 6, 7, & 8, Linux Mint 14, 17 & 18, FreeBSD 8.3, 10.3 & 11, pfSense 2.x, Mac OS X, Android using termux, Windows 10 bash, and msys / msys2 & cygwin environments.

### 1.3.3 What osync cannot do

Contrary to most sync tools, osync is stateful, which means that it doesn't need an agent installed on computers it synchronizes to. This makes it gain some degree of fault tolerance with WAN links.

But this also has some drawbacks, there is a cornercase when a user deletes a file while the synchronisation is happening, the file won't get deleted on the other side because the sync is actually updating the directory states at the same time they get changed. Even if this happens only on very big deployments in some rare occasions because of the time osync takes to sync, there is still a chance that your deletion wouldn't make it. Generally speaking, the easiest way is to setup osync as a crontask and launch it by night in order to avoid any possible

The good point is, osync is built from ground up to keep your data safe, and there's no chance it will delete anything you didn't want.

osync is a simple bash script that relies on other tools like rsync.

Hence, it has some advantages and disavantages:

Advantages:

- Easily customisable
- Fast
- Agentless

Disavantages:

- There's no way to detect file moves. If you move a directory on replica A, osync will soft delete the directory on replica B and copy the new directory from replica A.
- There's no multi-master replication in osync V1. Hence, if you want to sync replicas A, B, and C using osync, you'll have to use one of the following schemas:

**Replicate 3-way with A as master**

Run the following tasks sequentially where A is the initiatior:

Replicate A & B
Replicate A & C
Replicate A & B

**Replicate 3-way with A & B as masters**

Run the following tasks from each system. Be sure they won't run concurrently (osync will detect that another replication is still running, and abort the current one):

Replicate A & B where A is the initiator
Replicate B & C where B is the initiator
Replicate C & A where C is the initiator

## 1.4 Why use osync

There are a lot of file sync tools out there, some probably better than osync depending on the use case.

osync has been basically written to be low bandwidth friendly, with resume options for unstable internet links (but it will also do great on a fiber link :)).

It has also been written to be a setup and forget tool, without any user interaction like manual conflict resolution.

osync also is one of the few tools that **support ACL synchronization**.

At last, osync consumes very little RAM and CPU ressources and is suitable from lower en hardware up to serveres.

## 1.5 How osync tries to resolve sync issues

Let's get back to the example above, where osync is used to sync two remote offices with users' home directories.

Now imagine a user uploaded 100GB of data, and the WAN link between local and remote systems can only handle 6GB/hour of data transfer.

Now if osync is scheduled every night at 10:00 pm, and it's configured to run for maximum 10 hours, it would stop at 6am, after having transferred 60GB.

Then, on the next day, it would transfer the remaining 40GB from 10:00 pm to about 3:30am.

Also, if you run sequential instances of osync (with osync-batch), one per user directory for example, you can decide how much time osync should spend per user. So if a user uploads too much data, and osync cannot finish the synchronization task for that user directory in a given timespan, it will stop that sync task and run next user synchronization task so every user sync task gets run, regardless of the amount of data. If there's time left, osync-batch reprograms the user sync task that has been stopped.

## 1.6 Naming in this document

osync's goal is to synchronize two directories, that can be hosted on the same computer or two different ones.

The computer that runs osync must have at least one of these two directories mounted, and will be called the *local system.*

The first directory to sync on the local system is called the *initiator replica*.

The other directory, called the *target replica* can be hosted on the *local system*, or on another computer which will be called the *remote system*. In that case, the *local system* will connect to the *remote system* through an ssh tunnel and synchronize both *initiator* and *target replicas*.

Any osync configuration file that gets executed is called an *osync task*.

In the following examples, a special user called *syncuser* is used for osync.

## 1.7 How osync solves sync conflicts

Conflict resolution is done automatically. When a file is modified on both replicas, osync compares the timestamps on both replicas and keeps the most recent file.

In the highly uncommon case where both files have the exact same timestamp, a configuration value called CONFLICT_PREVALANCE choses the which replica's file will be kept. By default, Initiator is chosen, unless specified otherwise in the config file.

The file that isn't kept is copied to ".osync_wordir/backups" directory of the replica by default, with its full path relative to the replica, unless specified otherwise by the CONFLICT_BACKUP configuration value.

After an amount of days set by CONFLICT_BACKUP_DAYS, which is 30 by default, the backed up file is deleted.

If CONFLICT_BACKUP_MULTIPLE is set to YES (disabled by default), multiple versions of the backed up files are kept, with a timestamp suffix like "2016.12.31-12.00.01".

# 2 Prerequisites

## 2.1 General packages

osync is a bash script that will work with **bash version 3.2 or better**, but will not run with ksh / tsh / csh or other shells. Usually bash comes with most linux distributions.

Here's a list of commands to install basic prerequisites on both local and remote machines.

Note that openssh-clients are only needed on local machine if connecting remotely and sshpass is only needed if dealing with password files instead of RSA keys.

The sudo package is only needed if you plan to run osync remotely with a standard user that gets sudo privileges.

Also, inotify-tools / fswatch is only needed if you plan to run osync in daemon mode (wherever inotify-tools is available).

- RHEL / CentOS

  -
    ```
    yum install rsync coreutils openssh-clients sshpass inotify-tools
    ```
- Debian / Ubuntu / Mint

  -
    ```
    apt-get install rsync openssh-client sshpass inotify-tools
    ```
- FreeBSD

  -
    ```
    pkg install bash rsync sshpass inotify-tools sudo
    ```
- MacOS X
  -

    -
      ```
      brew install rsync fswatch
      ```
- MSYS (use management interface to install tools)

    -
      ```
      msys-base msys-coreutils-ext msys-rsync procps openssh-client
      ```
- MSYS2

    -
      ```
      pacman -S rsync procps openssh-client
      ```
- Cygwin (use management interface to install tools)

    -
      ```
      rsync procps-ng openssh wget
      ```
- termux (Android / Busybox)

    -
      ```
      apk --no-cache add bash rsync openssh-client
      ```

Additionnaly, on termux, shebangs need to be modified to work with the system root. Example on how to get osync.sh working:

```
termux-fix-shebang osync.sh
```

### 2.1.1 Mail alert prerequisites

osync can send alert emails on warnings / errors.

- On standard linux distros, BSD, MacOSX and Windows 10 bash, osync relies on your system-wide configured mail sending support in order to send alerts. It will try to use mail / mutt & sendmail. Please make sure that your system is configured to allow outgoing mail. You may check for osync mail in your mail logs (example: /var/log/maillog on RHEL / CentOS).
- On android / busybox, you will need sendmail and openssl-tools if you plan to use mail encryption (TLS/SSL)

    -
      ```
      apt --no-cache add sendmail openssl-tools
      ```
- On Cygwin / MSYS2, mail support is done via mailsend (https://github.com/muquit/mailsend) or sendmail (http://caspian.dotconf.net/menu/Software/SendEmail/) which have to be executable path (%PATH% variable, or just place them in system32).

### 2.1.2 Daemon mode prerequisites

osync can run in daermon mode in order to detect file modifications. This mode works on Linux / BSD and MacOSX.

On Linux and BSD, please install the following package:

```
inotify-tools
```

On FreeBSD, you might want to increase kern.maxfiles tunable if you plan to use inotify-tools for applications that need to monitor activity of a lot of files. Modify /boot/loader.conf and add

```
kern.maxfiles="25000"
```

On MacOS X, you will need to install fswatch with brew:

```
brew install fswatch
```

## 2.2 Downloading osync

osync can be downloaded on the author's site (stable version) or on github (stable or latest dev snapshot).

Getting osync via author's site

```
$ wget http://netpower.fr/projects/osync/osync.v1.2.tar.gz
$ tar xvf osync.v1.2.tar.gz
```

Getting osync via github (remove the -b ”stable” if you want latest dev snapshot)

```
$ git clone -b "stable" https://github.com/deajan/osync
```

On Linux / BSD / Windows 10 bash, Once you downloaded osync, enter into the newly created folder and run the install script

```
$ bash ./install.sh
```

This will copy osync to /usr/local/bin and create /etc/osync with a test sync.conf file.

It will also copy daemon required files to /etc/init.d or /usr/lib/systemd/system and /etc/systemd/user depending on your distribution.

On MacOS X, msys, Cygwin and termux (Android), you may directly use osync.sh script.

Uninsalling is done by using

```
$ bash ./install.sh --remove
```

There is also an RPM package available for Fedora & CentOS, which will install all binaries to /usr/bin instead of /usr/local/bin in order to enforce good practices. Please note that when using the RPM packages, the binaries paths in this document shall be read as /usr/bin instead of /usr/local/bin.

## 2.3 File synchronization

File sync tasks don't need any special configurations. You only have to worry about your sync user having the filesystem permissions to read / write on both replicas.

A good way is to make your user member of the files' group that has full permissions.

Another way to achieve this is using ACLs if your filesystem supports them. You can add the following permissions for user "syncuser" on directory "/home/web". Setting a default rule will add rights on new files.

```
# setfacl -dRm u:syncuser::r-x /home/web
```

Be aware that ACLs are tricky and default unix permissions serve as mask for ACLs.

Make always sure you can read /write to both replicas with your sync user:

```
# su syncuser
$ cat /initiator/replica/test.file
$ touch /initiator/replica/othertest.file
```

Repeat that step for the target replica.

## 2.4 Time setup

WARNING: osync's conflict resolution relies on timestamps, so it is very important for all systems osync runs / syncs to, to have a reliable and common timesource.

Please consider setting up NTPD first before you plan to run osync.

## 2.5 Performing superuser sync

osync can be run as superuser on the remote side, which should always be avoided by granting the read / write permissions to a dedicated sync user to both replicas.

There are still some cases where osync needs to be run as superuser, especially when syncing system files.

In those cases, osync can be run as dedicated sync user and ask for sudo permissions (if system permits it) for specific commands, or directly run as root (which isn't recommended).

In order to be able to use the sudo command without having to enter a password, you'll need to modify remote system to allow the following commands to be run as superuser.

Use visudo to edit the sudoers file (or carefully edit /etc/sudoers yourself) and add the following

```
syncuser ALL= NOPASSWD:SETENV:/usr/bin/rsync,/usr/bin/bash
```

Please note that the SETENV parameter is required so osync can send it's local variables as remote environment variables (equals sudo -E).

You might check the right paths to your commands, especially on some FreeBSD environments where rsync might be in /usr/local/bin instead of /usr/bin (example to get path for rsync executable):

```
$ type rsync
```

You'll also need to disable requiretty in /etc/sudoers by adding the following line:

```
Defaults:syncuser   !requiretty
```

Once your standard sync user is granted to run what osync needs, you can enable sudo in osync's config file:

```
SUDO_EXEC=yes
```

You should be aware that there is a risk with having rsync command run as superuser. A user who can run rsync command as superuser can upload any file he wants to the system, including a tweaked /etc/sudoers or /etc/passwd file. Please read chapter 2.8↓ to secure your installation.

## 2.6 Remote sync

osync can perform local or remote synchronization tasks. For local sync, pelease refer to chapters 3.2↓, 3.4↓and 3.5↓.

Remote synchronization is done through an SSH tunnel. To be able to establish such a tunnel without having to enter a password, you'll have to generate a pair of private and public RSA keys.

The private part is kept by the computer that initiates the connection, the local system. The public part is kept on the remote system.

The following steps will be required to generate a ssh key:

Create a dedicated sync user and log in as that user on the local system to perform the following actions.

```
$ ssh-keygen -t rsa
```

This should create two files named ~/.ssh/id_rsa and ~/.ssh/id_rsa.pub

You should also create a dedicated sync user on the remote system.

Copy the public part of the RSA pair to the remote system with scp (replace 22 with your ssh port number if needed).

```
$ scp -p 22 ~/.ssh/id_rsa syncuser@remotesystem.tld:/home/syncuser/.ssh/authorized_keys
```

Make sure the file is only readable and owned by the syncuser on the remote system.

```
$ chmod 600 /home/syncuser/.ssh/authorized_keys
$ chown syncuser:root /home/syncuser/.ssh/authorized_keys
```

Now you should be able to login as "syncuser" on the remote system without any password. You can try to remotely login by entering the following on the local system:

```
$ ssh -p 22 syncuser@remotesystem.tld
```

Be aware that only the user that generated the ssh key can remotely log in.

You may optionnaly enhance remote login security by applying chapter 2.8↓ methods.

## 2.7 Mail transport agent

You should make sure your system can send emails so osync can warn you if something bad happens. osync will use mutt or mail command. Please make sure you can send a test mail with at least one of the following commands run by your sync user:

On Linux / BSD / MacOS / Windows 10 Bash

```
$ echo "your test message" | mutt -x -s "This is a test message" your@mail.tld
$ echo "your test message" | mail -s "This is a test message" your@mail.tld
```

On termux (Android)

```
$ echo -e"Subject:subject\r\nyour test message" |sendmail -f "sender@mail.tld" -S "smtp.yourisp.tld:25" -au"MySMTPUser" -ap"MySMTPP
```

Check your antispam if you don't get your message. If you still don't get your message, check your distributions documentation about the mail command.

If you run on windows cygwin / msys environment, please make sure you can launch mailsend.exe / sendemail.exe by adding it to the %PATH% variable (found here and here).

## 2.8 Enhancing remote system security

You may want to secure a password-less ssh access by removing non necessary services offered by SSH. Edit the file ~/.ssh/authorized_keys created earlier on the remote system and add the following line in the beginning of the file:

```
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty
```

Also, we may want to prevent any host except of our initiator replica system to passwordless connect. Add the following line:

```
from=*.my.initiator.replica.server.domain.tld
```

Your authorized_keys file should look like this:

```
from="*.mydomain.tld",no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty ssh-rsa yourkey== syncuser@host.tld
```

## 2.9 More security enhancing (not recommended unless you know what you are doing)

We may also restrict the ssh session to only a couple of commands we'll need. osync comes with a script called *ssh_filter.sh* that will only allow execution of commands osync needs. Once again edit your authorized_keys file and add the following.

```
command="/usr/local/bin/ssh_filter.sh SomeAlphaNumericToken9"
```

Your file should then look like this:

```
from="*.mydomain.tld",no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty,command="/usr/local/bin/ssh_filter.sh SomeAlp
```

Copy then the script ssh_filter.sh to /usr/local/bin on the remote system. Don't forget to make it executable and make it owned by root

```
# chmod 755 /usr/local/bin/ssh_filter.sh
# chown root:root /usr/local/bin/ssh_filter.sh
```

Now, only the commands send by osync including a specific remote token may be executed via the ssh tunnel. You may enable / disable the usage of sudo command by editing the following value in the ssh_filter.sh script as well as in the osync config file:

```
SUDO_EXEC=yes
```

The remote token can be set in the config file and must match the one setup in authorized_keys file.

```
_REMOTE_TOKEN=SomeAlphaNumericToken9
```

You may also set it on the fly in quicksync mode with the parameter

```
$ osync.sh --remote-token=SomeAlphaNumericToken9
```

## 2.10 Security for the paranoid (not recommended unless you really know what you are doing)

Executing rsync as superuser is a security risk. A way to prevent rsync usage allowing only a symlink to be executed. Thus, a attacker script using rsync would not work. This kind of security is called "security by obscurity" and should generally not be the only security process, but makes any attack harder. First, let's create a symlink to rsync called let's say o_rsync, on both local and remote systems.

```
# ln -s $(type rsync) $(dirname $(type rsync))/o_rsync
```

Now edit the osync config file and change the following value:

```
RSYNC_EXECUTABLE=o_rsync
```

# 3 Running osync

## 3.1 Basics

Everytime osync is run, it runs with an INSTANCE_ID.

This INSTANCE_ID helps identifying successive runs on replicas, and helps differentiate concurrent runs of the same initiator replica to different targets.

By default, the INSTANCE_ID in quicksync mode is "quicksync_task". This might be overriden via command-line or by using a configuration file.

Keep in mind that two osync instances should never share the same INSTANCE_ID.

## 3.2 Running osync in quicksync mode

You just osync to sync two local dirs like this:

```
$ ./osync.sh --initiator=/path/to/dir1 --target=/path/to/dir2
```

You also may want to sync a remote directory.

You may specify an alternate SSH port directly in the URI. When ommited, SSH port 22 is used.

Also, if not set, the default RSA key will be read from ~/.ssh/id_rsa

```
$ ./osync.sh --initiator=/path/to/dir1 --target=ssh://remoteuser@remotehost.com//path/to/dir2
$ ./osync.sh --initiator=/path/to/dir2 --target=ssh://remoteuser@remotehost.com:22//path/to/dir2 --rsakey=/home/user/.ssh/other_key
```

The following command-line options are quicksync specific:

Use an RSA key to connect remotely

```
--rsakey=/path/to/rsa.key
```

Use a password file instead of an RSA key (needs sshpass utility installed)

```
--password-file=/path/to/passwd
```

Set INSTANCE_ID

```
--instance-id="some-instance-name"
```

Skip deletions on initiator, target or both (valid values are: initiator or target or initiator,target)

```
--skip-deletion=initiator,target
```

Send alert emails (only works with Linux / BSD / Windows 10 bash / MacOSX in quicksync mode)

```
--destination-mails="my@mail.tld myother@mail.tld"
```

## 3.3 Setting environment variables

### 3.3.1 General environment variables

Quicksync mode actually can make use of most options available in config files.

You may set them on the fly as environment variable like

```
$ PRESERVE_ACL=yes ./osync.sh --initiator=/path/to/dir1 --target=/path/to/dir2
```

A list of config values is found in appendix.

### 3.3.2 Special environment variables

There is a set of special environment variables used mainly for debugging purposes.

Valid ones are:

Enable deubgging

```
_DEBUG=yes
```

Enable paranoid debugging (only works on debug_osync.sh build or bootstrap.sh)

```
_PARANOIA_DEBUG=yes
```

Increase / decrease sleeping time between process checks when osync waits for a process. Default value is 0.05. (x is an integer or floating point number)

```
SLEEP_TIME=x
```

Ignore unknown OS and execute regardless with default options.

```
IGNORE_OS_TYPE=yes
```

## 3.4 Running osync with a full blown configuration file

Running osync with a configuration will do just the same as in quicksync mode, except that you have much more control of what's going on.

A sample configuration file is called sync.conf and is included with osync. You may edit this file to fit your needs. Basically configuration files should go to /etc/osync.

Every option of the configuration file is explained in the appendix.

Once you've setup a file according to your needs, you may go for a test run.

```
$ ./osync.sh /etc/osync/my_sync.conf --dry --verbose
```

osync should enumerate which changes will be done on both sides.

If everything worked out right, you might process the actual sync process.

A full configuration file specifies a maximum execution delay. Initial sync tasks can take a huge amount of time depending on bandwidth between replicas, in that case you might add parameter –no-maxtime to your first sync run so execution time won't be enforced.

```
$ ./osync.sh /etc/osync/my_sync.conf --no-maxtime
```

Creating a regular sync scenario is quite simple as long as you don't schedule twice the same sync task in a shorter time span than your HARD_MAX_EXEC_TIME_TOTAL value. Just create a crontab entry and add parameter –silent so your local mailbox won't get filled up. Example, having a sync scheduled every hour in /etc/crontab

```
00 * * * * syncuser /usr/local/bin/osync.sh /etc/osync/your_sync.conf --silent
```

Please note that this syntax works on RedHat / CentOS. On Debian you might need to remove the usename "ie syncuser" in order to make the crontab line work.
You may find the sync log under /var/log/osync-your_sync.log or under the current directory if /var/log is not writable.

## 3.5 Running osync as deamon

### 3.5.1 Manually

osync may also run in file monitor mode. In this mode, osync checks the initiator replica, and runs a synchronization as soon as there is file activity on initiator replica. With this mode, you do not need a schedule anymore. Be aware that only initiator replica is monitored, and target replica sync updates only occur when initiator replica modifications happen.

```
$ ./osync.sh /etc/osync/my_sync.conf --on-changes
```

### 3.5.2 As a system service

If you plan to run osync on a regular basis in file monitor mode, you might consider installing it as a system service.

From the directory you downloaded osync, run the install.sh script and enable the service.

Remark: When exiting osync daemon, the process will continue to run for up to a minute to unlock replicas and termine sub processes.

**init.d service files**
For init.d systems, syntax is:

```
# service osync-srv start
# chkconfig osync-srv on
```

osync then scans for *.conf files in /etc/osync and will run an instance per configuration file.

Service control just works like with standard system services.

**systemd service files**
For systemd systems, syntax is:

```
# systemctl start osync-srv@config_file
# systemctl enable osync-srv@config_file
```

With systemd, every config file found in /etc/osync can be controlled as a separate service.

Note that on FreeBSD, rsync exclusion patterns won't work because if inotifywait's limited –exclude implementation that can only make use of one exclude parameter.

## 3.6 Modifying osync behavior

### 3.6.1 Output / log behavior

On most runs, you may only want to output file modifications or errors. You may achieve this (in quicksync / conf / daemon modes) by adding the following parameters:

```
$ ./osync.sh /path/to/sync.conf --errors-only --summary --no-prefix
```

The following is a list of all command line switches that work with all three modes:

Will run a simulation only

```
--dry
```

Silents osync totally, to be used in a cron schedule

```
--silent
```

Show a summary of updated / deleted files

```
--summary
```

Suppress non error state messages

```
--errors-only
```

More detailled output (debug)

```
--verbose
```

Suppress log time prefix

```
--no-prefix
```

Add rsync stats to output

```
--stats
```

### 3.6.2 Execution behavior

Disable HARD_MAX_EXEC_TIME checks, so that a synchronization can take as long as it wants. This is used to perfom initial sync operations on big sets of files

```
--no-maxtime
```

Override any existing active or dead locks on replicas. Be careful if there is another running sync on any of both replicas

```
--force-unlock
```

Invoke daemon like mode. Shouldn't actually be used unless for debugging. Use daemon service files instead.

```
--on-changes
```

Show help

```
--help
```

## 3.7 Running osync batch

If you have multiple configuration files in /etc/osync that you would like to run sequentially, and re-run failed sync tasks, osync comes with a tool called osync-batch.

It will execute all osync conf files in a row, in a given timespan.

osync-batch passes all its parameters to osync itself, except for the following which modifiy batch behavior itself:

Path to directory containing osync configuration files

```
--path=/etc/osync
```

Maximum number of runs for failing synchronisation tasks (x is an integer). Defaults to 3.

```
--max-runs=x
```

Maximum execution time for whole batch process in seconds (x is an integer). Defaults to 36000. Setting to 0 disables max execution time check.

```
--max-exec-time=x
```

You may program a cron task for osync-batch.sh like

```
00 * * * * syncuser /usr/local/bin/osync-batch.sh --path=/etc/osync --silent
```

# 4 System & FS Caveats

Depending on which configuration osync runs, not all functions may work.

Especially, ACL and entended attributes preservation won't work on MacOSX, Cygwin, MSYS, Android and BusyBox local or remote environments.

In order to work on most systems, ACL preservation and extended system attribute preservarion has been disabled in quicksync mode.

To enable them, please run osync with the following environment variables

```
$ PRESERVE_ACL=yes PRESERVE_XATTR=yes ./osync.sh
```

Those settings are also valid if set in config files.

Also, setting up a sync between different file systems may prevent certain functionnality to work (mainly ACL preservation, extended attribute preservation and symlink / hardlink support).

At last, MSYS2 has symlink support disabled by default. It can be enabled by uncommenting MSYS=winsymlinks:nativestrict in msys2-shell.cmd, but doesn't behave exactly like other implementations.

# 5 Configuration file appendix

## 5.1 Full list of configuration file parameters

Set this to whatever you want to identify your sync task. Two different osync runs should never share the same INSTANCE_ID. This value also determines the log filename and appears in the warning / error mails.

```
INSTANCE_ID=name_of_your_sync
```

Initiator directory to sync (initiator replica), must be on the system you're running osync on.

```
INITIATOR_SYNC_DIR="/some/path"
```

Target directory to sync (target replica), can be on the same system you're running osync on or another remote system, reachable via an SSH tunnel.

Target directory can be a SSH uri like "ssh://user@host.com:1234//some/other/path" where 1234 is an optional port, and the first slash is a separator, meaning that the full path is /some/other/path.

```
TARGET_SYNC_DIR="/some/other/path"
```

Location of the private RSA key. If left empty, the default path "~/.ssh/id_rsa" will be used.

```
SSH_RSA_PRIVATE_KEY=~/.ssh/id_rsa
```

Alternate auth method by using a password file. This needs sshpass to be installed.

```
SSH_PASSWORD_FILE=/home/syncuser/path/to/passwd
```

When using ssh_filter security, you need to specify a remote token matching the one setup in remote authorized_keys file

```
_REMOTE_TOKEN=SomeAlphaNumericToken9
```

Tells osync to create initiator or target directories if they don't exist. Default is no.

```
CREATE_DIRS=yes|no
```

By default, leaving this empty sets the log file to /var/log/osync_INSTANCE_ID.log or ./osync_INSTANCE_ID.log if /var/log is not writable. You might change this to specify a personalized log file.

```
LOGFILE=""
```

Generate an alert if initiator or target replicas have less space than the following given value in kilobytes.

```
MINIMUM_SPACE=10240
```

Bandwidth limit in kilobytes / second. Leave this to zero to disable limitation. Note that bandwidth is enforced only when running file updates. All other sync steps don't enforce that setting as they generally need very little bandwidth.

```
BANDWIDTH=0
```

Synchronization tasks may be executed as root if you enable the following parameter. See prerequisites in chapter 2.5↑.

```
SUDO_EXEC=yes|no
```

Paranoia option. Don't change this unless you read chapter 2.10↑ and understand what you are doing.

```
RSYNC_EXECUTABLE=rsync
```

Remote Rsync Executable path. Don't change this unless your remote rsync binary isn't in the execution path.

```
RSYNC_REMOTE_PATH=""
```

Rsync include / exclude order. If set to include, includes will be processed before excludes, and vice-versa.

```
RSYNC_PATTERN_FIRST=include|exclude
```

List of files / directories to include / exclude from both replicas (see rsync patterns for more explanations, wildcards won't work).

Poaths are relative to both replicas. List is separated by PATH_SEPARATOR_CHAR defined below.

```
RSYNC_INCLUDE_PATTERN=""
RSYNC_EXCLUDE_PATTERN="tmp;archives;somepath"
```

File that contains the list of files /directories to include / exclude from both replicas (see rsync pattern files for more explanations). Leave this empty if you don't want to use an exclusion file. This file has to be in the same directory as the config file. Paths are relative to sync dirs. One element per line.

```
RSYNC_INCLUDE_FROM=""
RSYNC_EXCLUDE_FROM="exclude.list"
```

Path separator char for RSYNC_EXCLUDE_PATTERN, you might change this in the unholy case that your filenames contains semicolons.

```
PATH_SEPARATOR_CHAR=";"
```

Enable / disable ssh compression. Leave this enabled unless your connection to remote system is high speed (LAN)

```
SSH_COMPRESSION=yes|no
```

Tell ssh to not check the remote computer ssh fingerprint. DANGER WILL ROBINSON ! This should generally lead to security issues. Only enable this if you know exactly what you are doing.

```
SSH_IGNORE_KNOWN_HOSTS=yes|no
```

Ping remote host before launching synchronization. Be sure the host is responding to ping. Failing to ping will skip current task.

```
REMOTE_HOST_PING=yes|no
```

- Check for internet access by pinging one or more hosts before launching remote sync task. Leave this empty do disable the checks. Failed checks won't abort current task, just trigger a warning.

```
REMOTE_3RD_PARTY_HOST="www.kernel.org www.google.fr"
```

- Misc settings

Optional arguments to pass to rsync. Do not use already managed parameters by rsync (-r -l -p -t -g -o -D -E - u- i- n --executability -A -X -L -K -H -8 -zz –skip-compress –checksum –bwlimit –partial –partial-dir –no-whole-file –whole-file –backup –backup-dir –suffix --exclude --exclude-from --include --include-from --list-only --stats)

```
RSYNC_OPTIONAL_ARGS=""
```

Should osync preserve standard unix permissions

```
PRESERVE_PERMISSIONS=yes|no
```

Should osync preserve file owner

```
PRESERVE_OWNER=yes|no
```

Should osync preserve group owner

```
PRESERVE_GROUP=yes|no
```

Should osync preserve files executability status

```
PRESERVE_EXECUTABILITY=yes|no
```

Preserve ACLs. Please check that your filesystem supports ACLs and is mounted with it's support or rsync will get you loads of errors.

```
PRESERVE_ACL=yes|no
```

Preserve Xattr. The same applies as for ACLs

```
PRESERVE_XATTR=yes|no
```

Transforms symlinks into referent files/dirs when syncing replicas.

```
COPY_SYMLINKS=yes|no
```

Treat symlinked dirs as dirs. CAUTION: This also follows symlinks outside of the replica root.

```
KEEP_DIRLINKS=no
```

Preserve hard links. Make sure source and target FS can manage hard links or you will lose them.

```
PRESERVE_HARDLINKS=yes|no
```

Do a full checksum on files instead of comparing file sizes and modification times. Enabling this will make sync tasks longer.

```
CHECKSUM=yes|no
```

Use rsync compression for file transfers. Leave this disabled unless your're not using SSH compression.

```
RSYNC_COMPRESS=yes|no
```

Maximum execution time (in seconds) for sync process. Soft value generates a warning only. Hard value generates a warning and stops sync task.
You may set this to 0 to disable time checks.

```
SOFT_MAX_EXEC_TIME_FILE_TASK=7200
HARD_MAX_EXEC_TIME_FILE_TASK=10600
```

Log a status message every KEEP_LOGGING seconds. Set this to zero to disable status messages.

```
KEEP_LOGGING=1801
```

Minimum time (in seconds) in file monitor /daemon mode between modification detection and sync task in order to let copy operations finish.

```
MIN_WAIT=60
```

Maximum time (in seconds) in file monitor / daemon mode. After this amount of time, a sync operation is forced.

```
MAX_WAIT=7200
```

- Conflict and deletion option

Enabling this option will keep a backup of a file on the target replica if it gets updated from the source replica. Backups will be made to .osync_workdir/backups

```
CONFLICT_BACKUP=yes|no
```

Keep multiple backup versions of the same file. Warning, This can be very space consuming.

```
CONFLICT_BACKUP_MULTIPLE=yes|no
```

osync will clean backup files after a given number of days. Setting this to 0 will disable cleaning and keep backups forever. Warning: This can be very space consuming.

```
CONFLICT_BACKUP_DAYS=30
```

If the same file exists on both replicas, newer version will be synced. However, if both files have the same timestamp but differ, CONFLICT_PREVALANCE sets winner replica.

```
CONFLICT_PREVALANCE=initiator|target
```

On deletition propagation to the target replica, a backup of the deleted files can be kept. Deletions will be kept in .osync_workdir/deleted

```
SOFT_DELETE=yes|no
```

osync will clean deleted files after a given number of days. Setting this to 0 will disable cleaning and keep deleted files forever. Warning: This can be very space consuming.

```
SOFT_DELETE_DAYS=30
```

Deletion propagation (not soft deletion) can be skipped on one or both replicas.

```
SKIP_DELETION=initiator|target|initiator,target
```

- Resuming options

Try to resume an aborted sync task

```
RESUME_SYNC=yes|no
```

Number maximum resume tries before initating a fresh sync.

```
RESUME_TRY=2
```

When a pidlock exists on target replica that does not correspond to initiator's instance-id, force pidlock removal. Be carefull with this option if you have multiple initiators.

```
FORCE_STRANGER_LOCK_RESUME=no
```

Keep partial uploads that can be resumed on next run. This can be very useful if big files must get updated though slow links.

```
PARTIAL=no
```

Use rsync delta copy algorithm. Disabling this is useful to reduce CPU usage and raise bandwidth which might be valuable on local-local syncs or LAN syncs.

```
DELTA_COPIES=yes
```

- Alert Options

Optional mail body encoding (using iconv).
By default, all mails are sent in UTF-8 format without header (because of maximum compatibility of all platforms). You may specify an optional encoding here (like "ISO-8859-1" or whatever iconv can handle)

```
MAIL_BODY_CHARSET=""
```

List of alert mails separated by spaces

```
DESTINATION_MAILS="your@alert.tld"
```

MSYS / Cygwin / Android / Busybox only mail options

```
SENDER_MAIL="alert@your.system.tld"
SMTP_SERVER=smtp.your.isp.tld
SMTP_PORT=25
SMTP_ENCRYPTION=tls|ssl|none
SMTP_USER=optional_smtp_user
SMTP_PASSWORD=optional_smtp_password
```

- Execution hooks

Commands can will be run before and / or after sync process (remote execution will only happen if REMOTE_SYNC is set). Multiple commands can be semicolon separated.
Command(s) to run locally before sync process starts.

```
LOCAL_RUN_BEFORE_CMD=""
```

Command(s) to run locally if sync process finishes.

```
LOCAL_RUN_AFTER_CMD=""
```

Command(s) to run on remote system before sync process starts.

```
REMOTE_RUN_BEFORE_CMD=""
```

Command(s) to run on remote system if sync process finishes.

```
REMOTE_RUN_AFTER_CMD=""
```

Max execution time of commands before they get force killed. Leave 0 if you don't wan't this to happen. Time is specified in seconds.

```
MAX_EXEC_TIME_PER_CMD_BEFORE=0
MAX_EXEC_TIME_PER_CMD_AFTER=0
```

Stops osync execution if one of the above commands fail

```
STOP_ON_CMD_ERROR=yes|no
```

Run local and remote commands after a sync task even if it failed.

```
RUN_AFTER_CMD_ON_ERROR=yes|no
```

## 5.2 Upgrades from v1.0x and v1.1x

Upgrades from osync v1.0x require to rewrite osync config file and state filenames. Master & slave replica are now called initiator and target replicas. SYNC-ID is now called INSTANCE-ID.
There's an update script that does this job.
For quicksync tasks:

```
$ upgrade-v1.0x-v1.2x.sh --master=/path/to/master/replica --slave=/path/to/slave/replica --sync-id=[INSTANCE_ID]
```

For config file tasks:

```
$ upgrade-v1.0x-v1.2x.sh /etc/osync/sync.conf
```

Upgrades from osync v1.1x require to add osync configuration values into config file.
Just use the upgrade script the same way you would to upgrade from v1.0x.

# 6 Troubleshooting

osync has been tested successfully on multiple systems for a wide variety of sync plans. Please check the following steps before requesting help.
If no error message is shown in console, please check the log files.

## 6.1 Log files

Default [log_path] is /var/log, /home, ./ and /tmp depending if the directories are writable.
Everything that is printed on screen via stdout or stderr is always included in log file which is named osync.[INSTANCE_ID].log
Disabling switches like –errors-only or –silent may help diagnostic the problem.
Log inludes a prefix (normally TIME) which indicates since how much seconds osync runs.

When some commands get run remotely it will indicate how much seconds osync runs a specific remote task, beginning from 0 at every remote task.
Prefix can be disabled via a command line argument.

## 6.2 Warnings, errors and alerts

osync has two levels of alerts.
Warnings are raised when something minor goes wrong, like checking for minimum available space or checking for internet connectivity, which will not affect the synchronization task directly.
Errors are raised when something happens that could possibily stop sync from happening.
Both are logged, and if email support is set, an alert email will be sent.

## 6.3 Local-local sync

osync logs every of it's actions to [log_path]/osync-version.instance_id.log.
Please check the log file if something went wrong.
You might try running osync as root to check if your problem is filesystem permission related.
You might add –verbose option to see what actually happens.
Also, running osync with the following command will give the exact commands that actually happen:

```
$ DEBUG=yes /usr/local/bin/osync.sh /etc/osync/my_sync.conf --verbose
```

## 6.4 Local-remote sync

Remote synchronization is a bit more tricky.
You might check that you can log in remotely with the command

```
$ ssh -p 22 remotesyncuser@remotehost.tld
```

Also, you might check that you can use rsync command remotely

```
$ ssh -p 22 remotesyncuser@remotehost.tld rsync --help
```

You can temporarily disable ssh security by removing lines you added in chapter 2.8↑. Additionnaly, you can check ssh_filter log in ~/.ssh/ssh_filter.log on the remote system. You might try running osync with SUDO_EXEC to check if your problem is user permission related.

## 6.5 File monitor mode

In file monitor mode, osync will still log it's execution to /var/log/osync.instance_id.log (or current directory if /var/log is not writable).
Also, standard systemd log method is used if available. You may check an execution with

```
# systemctl status osync-srv@configfile
```

You mai also see osync logs in journalctl with

```
# journalctl -xn
```

# 7 Final words

The idea of osync came in a discussion around a beer one evening. It began as a project for a friend, whose company I was working for as a consultant.
Today, osync is still used by this company, and a lot of others around the globe.
I do provide technical help and support in my spare time, and will appreciate every contribution on Github :)