osync v1.1 Documentation

27 July 2016

http://www.netpower.fr/osync

Contents

1 Introduction

1.1 Quickstart guide

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

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, or totally restart the sync task if resume fails.

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

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 free disk space.

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 succesfully tested on RHEL / CentOS 5, CentOS 6, Centos 7, Debian 6, Debian 7, Linux Mint 14, 17, FreeBSD 8.3, 10.3, pfSense 2.x, Mac OS X, and Windows using msys & cygwin environment.

1.3.3 What osync cannot do

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

Hence, it has some advantages and disavantages:

Advantages:

- It's easily customisable
- It's fast

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

Hence, it has some unique features that make it a good tool depending on the use case.

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 programmed in bash and will not run with ksh / tsh or other shells. Usually bash comes with most distributions.

On FreeBSD, you might need to install bash with

```
pkg install bash
```

The following packages are needed on both local and remote systems:

```
rsync coreutils
```

Also, the local system will send emails on errors.

Make sure you have a mail package like mailx, mutt, postfix installed on Unix like systems.

On Winows, make sure you have mailsend (https://github.com/muquit/mailsend) / sendmail (http://-caspian.dotconf.net/menu/Software/SendEmail/) installed in executable path.

Additionnaly, if you intend to run osync in daemon mode, you'll need the following package.

```
inotify-tools
```

On MinGW, you will have to install msys-rsync and msys-coreutils-ext on top of a base install.

2.2 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.

2.3 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.

2.4 Performing superuser sync

osync may be run as superuser, 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 for specific commands.

In order to be able to use the sudo command without having to enter a password, you'll need to modify the local and / or remote system to allow the following commands to be run as superuser: rsync, du, find, mkdir, rm, echo, mv, tee and cat.

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

```
syncuser ALL= NOPASSWD:/usr/bin/rsync
syncuser ALL= NOPASSWD:/usr/bin/du
syncuser ALL= NOPASSWD:/bin/find
syncuser ALL= NOPASSWD:/bin/mkdir
syncuser ALL= NOPASSWD:/bin/rm
syncuser ALL= NOPASSWD:/bin/mv
syncuser ALL= NOPASSWD:/bin/echo
syncuser ALL= NOPASSWD:/bin/cat
syncuser ALL= NOPASSWD:/usr/bin/tee
```

You might check the right paths to your commands (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 2.7 to secure your installation.

2.5 Remote sync

osync can perform local or remote synchronization tasks. For local sync, pelease refer to chapters 3.2, 3.3and 3.4.

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 2.7 methods.

2.6 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:

```
$ 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
```

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

2.7 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
```

2.8 More security enhancing

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"
```

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" ssh-rsa yourkey==
    syncuser@remotesystem.tld
```

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 "find, du, rsync, echo, mv, mkdir and sudo" 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:

```
SUDO_EXEC=yes
```

Also, adding remote pre- and postexecution commands in your configuration files will not work if you use the ssh filter. You'll have to add your optional commands in ssh_filter.sh. Example if you want to perform remote snapshots you'll have to allow one of the following:

```
CMD1=zfs
CMD2=xfs
CMD3=lvm
```

2.9 Security for the paranoid

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 ssh_filter.sh and change the following value:

```
RSYNC_EXECUTABLE=o_rsync
```

Also, edit RSYNC_EXECUTABLE value on any of your sync configuration files and you're done.

3 Getting and running osync

3.1 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.v1.1.tar.gz
tar xvf osync.v1.1.tar.gz
```

Getting osync via github (remove the -b "v1.00a" if you want latest dev snapshot)

```
git clone -b "v1.1" https://github.com/deajan/osync
```

Once you downloaded osync, enter into the newly created folder and run the install script

```
sh ./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.

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
```

3.3 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
```

You may find the sync log under /var/log/osync-your_sync.log or under the current directory if /var/log is not writable.

3.4 Running osync as deamon

3.4.1 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
```

3.4.2 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.

3.5 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 alphanumerical order, in a given timespan.

osync-batch takes the following non mandatory parameters:

–silent Will launch osync tasks silently

–dry Will launch osnyc tasks as simulations only

–verbose Will launch osync tasks with detailed output, including changed and deleted files lists on both sides

–no-maxtime Will launch osync taks without any maximum execution time

–path= By default, osync-batch.sh searches for config files in /etc/osync. This parameter overrides the default value.

–max-reruns= By default osync-batch.sh tries to rerun failed tasks 3 times. This parameter overrides the default value.

–max-exec-time= By default, osync-batch.sh won't launch next task if 36000 seconds have passed. This parameter overrides the default value.

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

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

4 Configuration appendix

4.1 Quicksync only command line parameters

–initiator="" Initiator replica path. Will contain state and backup directory (is mandatory)

–target="" Local or remote target replica path. Can be a ssh uri like ssh://user@host.com:22//path/to/target/replica (is mandatory)

–rsakey Alternative path to rsa private key for ssh connection to target replica (if not ~/.ssh/id_rsa)

–instance-id Optional sync task name to identify this synchronization task when using multiple targets

4.2 Universal command line parameters

When run without any parameter, osync will show usage.

Both quicksync and config file modes can take the following optional parameters:

–dry Will make osync run a simulation only

–silent Will run osync silently, to be used in a cron schedule

–verbose Will run osync with detailed output, including changed and deleted files lists on both sides

–stats Will add rsync transfer statistics to verbose output

–partial osync will leave partial transfered files in order to be resumed on later runs

–no-maxtime Will disable MAX_EXEC_TIME checks, so a task can take as long as it needs. This is useful for performing initial big sync operations

–force-unlock Will override any existing active or dead locks on initiator and target replica

–on-changes Will launch a sync task after a short wait period if there is some file activity on initiator replica. You should try daemon mode instead

–help Will print osync version and usage

4.3 Full list of configuration file parameters

Set this to whatever you want to identify your sync task. 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
```

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. 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.

```
BANDWIDTH=0
```

Synchronization tasks may be executed as root if you enable the following parameter. See prerequisites in chapter 2.4.

```
SUDO_EXEC=yes|no
```

Paranoia option. Don't change this unless you read chapter 2.9 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.

```
REMOTE_RSYNC_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 check. Failing to ping will skip current task.

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

* Misc settings

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
```

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=300
```

* 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
```

* 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
```

* Alert Options

List of alert mails separated by spaces

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

Windows (MSYS / Cygwin environment) only mail options (used with mailsend.exe from muquit or sendemail.exe from Brandon Zehm)

```
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 Troubleshooting

osync has been tested successfully on multiple systems for a wide variety of sync plans. Please check the following steps before requesting help.

5.1 Local-local sync

osync logs every of it's actions to /var/log/osync-version.instance_id.log (or current directory if /var/log is not writable).

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
```

5.2 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 2.7. 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.

5.3 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
```

6 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 i get on Github :)