# rsync daemon service on OpenIndiana

⚠️ As of December 2011, OpenIndiana still does not include a service manifest for running an `rsync` daemon out of the box. Until this changes, others may find the following service manifest and method files helpful.

This has been tested in OpenIndiana 151a.

## Install and configure the SMF service for rsync and the daemon itself

Before starting with this guide, make sure that `rsync` software is installed on the machine. It is not installed per default in local zones, but can be done so with `pkg`:

```
pkg install network/rsync
```

## Prepare files for SMF services configuration

*To see all attachments described in this page, click on the small paper clip icon at the top left.*

Add SMF manifest and method scripts, and initial configuration for `rsync` itself (as `root`):

- copy (or `wget`) the attached file "rsyncd.xml" to: `/var/svc/manifest/network/rsyncd.xml`
- copy (or `wget`) the attached file "rsyncd" to: `/lib/svc/method/rsyncd`
- set the execute permission on the above file for user `root`:

  ```
  chmod u+x /lib/svc/method/rsyncd
  ```

  *(Thanks to* http://robbiecrash.me/?p=22 for pointing out that I missed the above `chmod` step)

- copy (or `wget`) the attached file "rsync" to: `/etc/default/rsync`
  NOTE that the variable `RSYNC_ENABLE` is already set to true in this file.
- now create a valid `/etc/rsyncd.conf` file. This must be done or the service will not run.
  For a full list of options see: `man rsyncd.conf`

  For an example see this trivial `/etc/rsyncd.conf` file:

```
use chroot = yes
read only = yes
log file = /var/adm/rsyncd_upload.log
log format = - %a - %f
transfer logging = yes

#module to share ISO files
[ISO]
path = /lift/data/ISO
comment = public ISO repository
```

## Import the SMF service manifest

Now you can import the SMF service manifest with the following code:

```
svccfg -v import /var/svc/manifest/network/rsyncd.xml
```

Next check to see the status of the `rsyncd` service – it should be defined but may be not running yet:

```
svcs -a | grep rsyncd
```

...you should see something like this:

```
offline         2:04:55 svc:/network/rsyncd:default
```

## Enable the SMF service

Now you need to enable the service:

```
svcadm enable network/rsyncd
```

## Test that the service is enabled and the daemon actually runs

Now execute the `svcs` command again and also execute `ps`:

```
svcs -a | grep rsync
```

The service should be listed as `online` now:

```
online         May_16   svc:/network/rsyncd:default
```

Now execute:

```
ps aux | grep rsync
```

you should see something like this

```
root    13351  0.0  0.0 2688 1340 ?        S 02:04:55  0:00 /usr/bin/rsync --d
```

## Test from a client

Note – if you add new modules to `/etc/rsyncd.conf` you should not have to restart the `rsyncd` service, as the file is read each time an `rsync` client connects.

Congrats, your `rsyncd` should be working now. Test it out from a client machine using a small file for testing with something like:

```
rsync servername::ISO/some_file.iso /localpath/here/
```

or just get a list of the files and/or directories by using verbose mode and piping to `less`:

```
rsync -v servername::ISO | less
```

Consult the log file referenced in your `rsyncd.conf` file to see info on file transfers. (In this example `/var/adm/rsyncd_upload.log`).

Thanks to Marcelo Leal for the rsyncd method file (changed here to point to `/usr/bin/rsync`) and also for parts of the service manifest.

# Optional – add automatic ZFS snapshots after rsync uploads

You might want to create ZFS snapshots linked to your incoming `rsync` upload jobs – for example, after a machine backing up to your OpenIndiana server disconnects its rsyncing session. Tricks like this can be done by defining a `post-xfer-exec` hook pointing to a script which handles the logic, in this case rsync-zfshot.sh.

Expanding on a trivial example above, here is another one:

```
use chroot = yes
read only = yes
log file = /var/adm/rsyncd_upload.log
log format = - %a - %f
transfer logging = yes
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsyncd.lock

#module to share ISO files
[ISO]
path = /lift/data/ISO
comment = public ISO repository

#Module available only to a multihomed host which backs up into this path
#and chowns the files to specified UID:GID and sets FS access rights
#(source is a Windows machine with separate user namespace); it is also
#recommended that each backed-up host has a dedicated backup ZFS dataset.
#When a backup attempt is complete, a ZFS snapshot is created, so we can
#walk back to any given complete backup image (i.e. via .zfs/snapshots
#directory with Samba/kCIFS sharing, or via MS Shadow Volumes with kCIFS)
[backup-jamais]
    path = /export/DUMP/windows/JAMAIS
    read only = no
    auth users = backup-jamais
    charset = UTF-8
    ### NOTE: cwRsync clients might require "--iconv=cp1251" on their command
    ###       line to properly upload russian-named files from Windows
    uid = backup-jamais
    gid = backup-jamais
    incoming chmod = Dug=rwX,o-rwx,Fug=rw,o-rwx
    hosts allow = 192.168.1.101/32 123.45.68.101/32
    post-xfer exec = /root/rsync-zfshot.sh -w -v -l -ok
```

The main trickery for this particular quest is in the rsync-zfshot.sh script attached to this post. It has a number of command-line options to control logging and verbosity, as well as to enable actual snapshotting (dry-runs by default), and perhaps only when the rsync upload was successful (without an rsync error status set). Hopefully, the code and comments (and command-line help) are its best documentation.

- copy (or `wget`) the attached file "rsync-zfshot.sh" to: `/root/rsync-zfshot.sh` or any other location referenced in your config file module above
- set the execute permission on the above file for user `root`:

```
chmod u+x /root/rsync-zfshot.sh
```

- You might be required to also delegate the ZFS snapshooting access rights to the backup user (but test first to see if this is really needed):

```
zfs allow -l -d -u backup-jamais snapshot,rename,mount \
  rpool/export/DUMP/windows/JAMAIS
```

NOTE: the `-l` `-d` options set this permission **l**ocally on the named dataset and on its **d**escendants, including those created in the future.