Incremental Backup Script

Jason Healy, Director of Networks and Systems

Last Updated Mar 18, 2008

Contents

Chapter 1

Incremental Backup Script

Last updated 2008/03/18

1.1 Introduction

Suffield Academy provides networked disk space for all of its users, and encourages its use as part of a regular backup strategy. We back up the entire disk array nightly to another machine as part of our disaster recovery plan.

Unfortunately, we do not have enough space to archive each of these full nightly backups. So, while we are protected against the server crashing, we do not have the ability to recover files from a particular point in time.

To remedy this problem, we designed a custom backup script to satisfy the following criteria:

- Perform "snapshot" backups that capture the entire file structure at that point in time.
- Only store changed data for each backup.
- Allow for exclusion of particular files (temporary, cache, trash, *etc.*).
- Allow for exclusion of directories that are over a certain allowed size (*e.g.*, 1 GB).
- Report excluded directories to the users that own the files.

We found that the best way to accomplish this was to use a collection of scripts to wrap existing utilities and provide the functionality we needed.

We use `rsync` as the tool to perform the backups, and it is the workhorse of our system. Rsync contains configurable settings to allow the exclusion or inclusion of particular files, and we can script the automatic exclusion of users who are over a particular quota.

Meanwhile, Rsync also includes the ability to hard-link against existing files (a method later used by Apple's Time Machine), so that you can store full file system copies, but only have to use disk space for the files that have changed. While not quite as good as copy-on-write filesystems that operate at the block level, this approach is very portable across operating systems.

Finally, we wrap Rsync in custom scripts that archive completed backups, timestamp them, and weed out older backups to conserve space.

The final collection of scripts are written in Perl and Bash, with very few module dependencies. The scripts include full documentation and are configurable.

1.2 Design Issues

(This section deals with the design considerations for the script. If you just want to start using the script, skip down to the usage section.)

While searching for methods to create *snapshot backups*, we found an excellent strategy for backing up only incremental changes. It involves using *hard links* on the filesystem to store redundant (*e.g.*, unchanged) files, and rsync to transfer only files that change between backup sessions. Please read the paper for more information; a discussion of the technique is beyond the scope of this document.

We investigated existing solutions that use this strategy (including a program called rsnapshot, which looked very promising. Based on the features and existing code base, we decided to adopt this strategy for our backups.

We would have liked to use an existing solution for backing up, but encountered a few significant issues that could not be resolved without writing custom code:

- Preservation of HFS metadata, such as resource forks and creator codes. Currently, only Rsync version 3 and up support all of the necessary metadata preservation that we need.
- Tight integration with our directory service to exclude users who are over quota from backup, and notifying these users automatically via e-mail.

To attain these goals, we use Rsync 3 as the core of the backup system, and wrap it with shell scripts that provide the additional policies that we need to back up the correct information and retain it.

In the sections below, we discuss each of the issues in more detail.

1.2.1 HFS Metadata

Most Macintosh computers running OS X use the HFS+ filesystem for storing data. HFS+ has two features which frustrate file transfers, especially to non-Mac OS X systems: **file metadata** (especially **type and creator codes**), which have no counterpart on other file systems, and **forked files**, which split files into several unique parts. Because other filesystems do not use this approach, storing HFS+ data correctly becomes much more difficult.

There is a great utility called Backup Bouncer that automatically tests backup tools and their ability to preserve these sorts of data. Based on our testing, we discovered that a patched build of Rsync 3 provided the best retention of metadata while also supporting a very efficient transfer mode (differential copies with hard-links at the target).

1.2.2 Directory Services Integration

To save space in our backups, we needed the ability to exclude files from the backup based upon a user's quota status. We do not enforce a hard quota limit on our fileserver (to allow users to store large files temporarily), but we didn't want to waste space with large files that didn't need to be backed up.

When backing up user home directories, the script communicates with our Open Directory server to find users that are over quota. If a user is over their quota, files are excluded from the backup (until the non-excluded files fit within their quota). When a user's files are excluded, their e-mail address is queried and the user is notified that certain files were not backed up.

1.2.3 Remote Backups

Rsync can perform backups via the network, and we have designed our scripts to allow this behavior as well.

Because our script performs various housekeeping duties (rotating directories, locating old directories to link against, *etc.*), remote backups must adhere to a specific way of doing things in order to work properly.

We do this by using Rsync's daemon mode, and using a pre/post execution script. The script is automatically run by rsync before and after the transfer, and it takes care of creating and renaming directories, ensuring that transfer options are set correctly, and other housekeeping. Because it's run on the server

side, the sender (client) simply uses regular rsync without needing to worry about the policies on the server.

1.3 Script Usage

The scripts depend on the following packages to work correctly:

rsync 3.0+ The rsync binary must be installed on the system somewhere. The name does not have to be "rsync"; you may simply change the name used in the configuration file. Note that if you are going to support Mac OS X extended attributes (metadata), you'll need to use a patched version of rsync.

To download and build your own version of rsync, follow these directions (many thanks to Mike Bombich and his tips posted at http://www.bombich.com/mactips/rsync.htm

First, download and unpack the sources (substitute the current version of rsync instead of "3.0.6"):

```
wget 'http://rsync.samba.org/ftp/rsync/rsync-3.0.6.tar.gz'
wget 'http://rsync.samba.org/ftp/rsync/rsync-patches-3.0.6.tar.gz'
tar -zxf rsync-3.0.6.tar.gz
tar -zxf rsync-patches-3.0.6.tar.gz
```

The final step above merges the patches into the main source tree.

Next, move into the source tree and apply the Mac OS X metadata patches:

```
cd rsync-3.0.6
patch -p1 <patches/fileflags.diff
patch -p1 <patches/crtimes.diff
```

Now configure the sources:

```
./prepare-source
./configure
```

And finally, build and install:

```
make
sudo make install
```

You should be all set with a patched version of rsync (installed to `/usr/local/bin` by default). Run `rsync --version` to confirm the version and patches.

Unix::Syslog.pm This is a reasonably standard perl module which allows the scripts to record debugging information directly to syslog on your system.

Rsync is run in "daemon mode" on the server you wish to use as the destination for backups. The daemon is normally started as root (so that it can bind to a privileged port), and then an alternate user is specified to own the tranfered files.

You can use the standard processes for starting a daemon on your system (init, daemontools, rc, etc). If you use Mac OS X as your server, we have Launchd plist files available for starting and running rsync. There are two files:

1. rsync-daemon.plist
2. rsync-watcher.plist

The first launches the rsync daemon, but only if the path where the backups go is present (we use an external disk for our backups, so we don't want the daemon to run if the disk isn't mounted).

The second watches the backup path and kills rsync if it is no longer available (the disk is unmounted). You'll need to customize the paths if you plan to use this functionality. If you don't need to worry about the path unmounting, you can just use the first script to keep the daemon alive.

Finally, Mac OS X users should ensure that the destination directory for their backups has **Ignore Ownership and Permissions** turned **OFF**. You can check this by choosing "Get Info..." on the destination volume. If privileges are not preserved, then rsync will assume that all the files have changed ownership (since, as far as it knows, they have), and every file will be retransmitted, making the backup non-incremental.

1.3.1 Organization

There are two main scripts to use as part of this system: one for the client (sender) and one for the server (receiver). Others are available for special tasks. Each script includes documentation in comment form, but we also have a brief introduction below:

`rsyncd_prepost` **(Download)** This file is executed by the rsync daemon on the server before and after every transfer. It also can be run from the command line to create the basic directory structure for a backup set.

`rsync_snapshot` **(Download)** This is a sample script to use on a client machine to send a backup to the server. It basically gathers up the username,

password, and paths, and then calls rsync. You may need to customize it for your client machines (for example, removing unused rsync flags, or adding exclude rules).

`users_over_quota` **(Download)** We use this script to total up the home directory sizes on our file server and write rsync excludes that eliminate large files from the backup. This script is custom to our environment, and you don't need it as part of the backups. We include it here so you can see how you might customize a backup using a pre-processing script that writes out a list of files to exclude from the backup.

1.3.2 Configuration Files

On the client side, there are no configuration files. All configuration is done directly when you invoke `rsync`, so you should modify the script that uses rsync to have whatever options/includes/excludes that you want.

On the server side, the `rsyncd_prepost` script shares the same configuration file as the `rsync` daemon (typically, `/etc/rsyncd.conf`). Our script parses the `rsyncd.conf` file to get the same options that rsync uses. Additionally, you can specify settings for the `rsyncd_prepost` script by placing special comment lines in the configuration file that our script understands (see below for an example).

Examples

We've included a sample `rsyncd.conf` file below. This file defines a single transfer target called "test". We've included detailed comments for each option so you know why the values are specified in this particular format.

Note: the rsync daemon re-reads its configuration file for every connection, so it is not necessary to signal (*e.g.*, HUP) the daemon if you change the config.

```
# /etc/rsyncd.conf


# This option forces rsync to write its PID out to a file.  Our
# launchd watcher script uses this PID to send signals to rsync, so
# this path should match that used by launchd.
pid file = /var/run/rsyncd_snapshot.pid

# Ordinarily, the user that starts the daemon process also owns all
# the files from the transfer.  We choose a non-root user here to own
# all the files.  The prepost script also uses these values when
# creating and setting ownership on directories.
#
# See also "fake super" and "incoming chmod" below
```

```
uid = rsync_user
gid = rsync_user

# Additional safety measure: change to the destination root of the
# transfer before executing any operations.
use chroot = yes

# use syslog instead of plain file logging
syslog facility = ftp

# enable plain file logging for more detail and debugging
#log file = /tmp/rsyncd.log
#transfer logging = yes
#log format = "%m:%u %h %o %f (%b/%l)"

# The following file contains the usernames and passwords for
# connections to the daemon.  File format is username:password, one per
# line.  We use this to restrict transfer modules to specific users.
secrets file = /etc/rsyncd.secrets

# Normally, to preserve ownership and permissions, rsync must run as
# root.  However, by using "fake super", rsync stuffs all file
# metadata into xattrs on the files and lets them be owned by a
# non-root user.  Additionally, this allows you to store metadata
# not supported by the native file system.
fake super = yes

# When using fake super, you are not running the transfer daemon as
# root.  This means that certain operations on the destination files
# can fail (such as setting attributes) if the permissions on the
# files are not permissive enough.  For example: if a file does not
# have read permission for "user", when it is transferred to the
# destination the rsync daemon user (who is not root) cannot read its
# xattrs to determine if the file changed, resulting in an error.
#
# To work around this, we set a user-permissive umask (or chmod) on
# the receiving side to guarantee that the rsync daemon can at the
# very least read and write to files it owns.  Be aware that when you
# restore files back to a host, the permissions may be (slightly) more
# permissive, though it's rare that you actually have a file that the
# owner cannot read or write to...
incoming chmod = u+rwX

# We specify our pre/post script for all transfers, using this file
# (rsyncd.conf) as the first argument so it can find all configuration
# options.
pre-xfer exec = /usr/local/bin/rsyncd_prepost /etc/rsyncd.conf
post-xfer exec = /usr/local/bin/rsyncd_prepost /etc/rsyncd.conf

# to prevent multiple connections overwriting each other, only allow
# one connection at a time (note that you must specify a lock file for
# each module, as the connection limit is enforced on a per-lock-file basis).
max connections = 1

# allow shares to be writeable (otherwise it's hard to back up to them!)
read only = no
```

```
# by default, don't advertise module names
list = no

# Comments that begin with "rsyncd_prepost" are read by the prepost
# script and used to set values in that script.  Because they are
# comments, they are ignored by the rsync daemon.

# date format for snapshot directory names (per strftime())
# rsyncd_prepost: dateformat=%Y-%m-%d

# The pre/post script renames backups by the date and time they
# complete.  Since backups take a variable amount of time to finish,
# it can be helpful to round off the time to the nearest hour/day/etc.
# Specify a time in seconds that you would like to round do (we
# use 1 day, or 86400 seconds).
# rsyncd_prepost: dateround=86400

# This is a module named "test".  In general, you want one module per
# set of files you're going to back up.  Most of our modules are for a
# single server, though some larger servers use multiple modules to
# spread out the organization of the files.
[test]
        comment = This is the comment

        # The path MUST always end in "rsync", and the level above
        # that should match the name of the module.  The pre/post
        # script will create the module dir and any necessary subdirs
        # if you run the prepost script with the module name as the
        # last argument.
        path = /Volumes/encrypted/test/rsync

        # To limit the number of connections to a particular module,
        # you must specify a lock file that is unique to that module.
        # Otherwise, the shared connection limit is global (for all
        # modules) and you'll likely get conflicts.
        lock file = /var/run/rsync_snapshot_test.lock

        # List any users from /etc/rsyncd.secrets that should have
        # access to this module
        auth users = test

        # List any machines that should have access to this module
        # (typically, only the machines that are sending the backups)
        hosts allow = 192.0.2.1

        # If desired, you can specify per-module options for the
        # pre/post script here as well.  The lines below define how long
        # snapshots are kept, and how far apart in time they are spaced.
        # See the pre/post script for more details.
        # rsyncd_prepost: snapshotpreserve=60 30
        # rsyncd_prepost: snapshotpreserve=120 60
        # rsyncd_prepost: snapshotpreserve=300 -1
```

1.4 Recovery

To recover files from the backup server, there are two basic options.

The first is to create a new module specification that includes the path to the snapshot directory you wish to recover from. You may specify an additional username or host restriction if you're recovering from a different host. Be sure to OMIT the "pre-xfer" and "post-xfer" lines from the config so the server doesn't think it's a live backup request.

However, that approach is cumbersome, and requires editing and saving files based on parameters at the time of recovery. An easier approach is to tunnel the rsync request via SSH and specify the path to recover from on the fly.

To do this, you need to have SSH access to the server, and that SSH user must have (at least) read permissions to the module folder you wish to restore from.

You can then use rsync tunneled through SSH to connect to the machine and restore the files. Consider the following example (which should be typed all on one line, or use continuation backslashes as we have here):

```
sudo /usr/local/bin/rsync -aNHAXv -e ssh \
--rsync-path="/usr/local/bin/rsync --fake-super" \
ssh_user@backups.example.com:/path/to/module/snapshot/2009-10-11/ \
/tmp/restore_to_here/
```

The rsync options (`-aNHAX`) should match those used when you created the backup from the sender. The `-e ssh` tells rsync to use the SSH tunnel instead of a direct connection. If your server normally has the **fake super** option set in its `rsyncd.conf` file, you need to tell the tunneled rsync daemon to turn it on as well using the `--rsync-path` option as we have. Finally, you specify the full source and destination path. In the case of the most recent backup, it will live in `.../module/rsync/complete/`. Otherwise, you can specify the snapshot to restore from with `.../module/snapshot/(date)/` as we have above.