Configuring Mac OS X for Unattended Backup Using rsync

Introduction

rsync is a command-line tool built-in to Mac OS X that allows you to synchronize files between two folders on a machine or between two machines on a network. rsync is an incredibly useful tool that has been used by Unix administrators for many years.

Focusing on the use of rsync between two separate machines, this article describes how to configure your two machines with host-based authentication to allow for unattended backups using rsync. The instructions that follow require extensive use of the Terminal, root access (via sudo), and an installation of Apple's Developer Tools.

Overview of an rsync session

1. Issue the rsync command with appropriate flags
2. rsync opens up an SSH connection to the server (even if it is the localhost)
3. The server asks your machine to authenticate. There are a few different ways to authenticate.
4. Once you have successfully authenticated, rsync runs on the server in server mode, sending the requested data to the instance of rsync on the client machine.
5. The connection is closed when the process has completed.

Overview of the setup process

On the client and server

I strongly encourage you to use rsync 3.0.6 or greater for Tiger and Leopard. The version of rsync that Apple ships with Tiger and Leopard does not perform as well as rsync 3.0.6, consumes more memory (especially for transfers of many files), and will copy unmodified resource forks every single time. This article will describe the use of rsync 3.0.6, therefore the command-line arguments may be incompatible with the Apple-supplied rsync.

1. Download the source to rsync 3.0.6 and compile it on both the client and server

On the client(s)

1. Create authentication keys on the client
2. Copy the client's public key to the server's authorized_keys file
3. Create a script to automate rsync
4. Create a LaunchDaemon to run the script at regular intervals

On the server

1. Create an rsync wrapper script to limit the access of the client
2. Modify the authorized_keys file to limit the client's access to the wrapper script only

Compile rsync 3.0.6

Follow these instructions in Terminal on both the client and server to download and compile rsync 3.0.6:

Download and unarchive rsync and its patches

```
cd ~/Desktop
curl -O http://rsync.samba.org/ftp/rsync/rsync-3.0.6.tar.gz
tar -xzvf rsync-3.0.6.tar.gz
rm rsync-3.0.6.tar.gz
curl -O http://rsync.samba.org/ftp/rsync/rsync-patches-3.0.6.tar.gz
tar -xzvf rsync-patches-3.0.6.tar.gz
rm rsync-patches-3.0.6.tar.gz
cd rsync-3.0.6
```

Apply patches relevant to preserving Mac OS X metadata

```
patch -p1 <patches/fileflags.diff
patch -p1 <patches/crtimes.diff
```

Configure, make, install

```
./prepare-source
```

```
./configure
make
sudo make install
```

Verify your installation

```
[bombich:~] /usr/local/bin/rsync --version
rsync version 3.0.6 protocol version 30
Copyright (C) 1996-2008 by Andrew Tridgell, Wayne Davison, and others.
Web site: http://rsync.samba.org/
Capabilities:
64-bit files, 32-bit inums, 32-bit timestamps, 64-bit long ints,
socketpairs, hardlinks, symlinks, IPv6, batchfiles, inplace,
append, ACLs, xattrs, iconv, no symtimes, file-flags

rsync comes with ABSOLUTELY NO WARRANTY. This is free software, and you
are welcome to redistribute it under certain conditions. See the GNU
General Public Licence for details.
```

By default, rsync will be installed in /usr/local/bin. If that isn't in your path, you will need to call your new version of rsync by its absolute path (/usr/local/bin/rsync).

Note: rsync may be updated more frequently than this article. Check the **rsync downloads page** for the latest version of rsync and update the instructions above appropriately. [Last updated 1/1/09]

Create authentication keys

First, determine if you already have authentication keys. In the Terminal, run:

```
sudo ls -la /var/root/.ssh
```

If you see "id_dsa" and "id_dsa.pub", then you can skip the rest of this section.

On the client machine, run the following in the Terminal:

```
sudo ssh-keygen -t dsa -f /private/var/root/.ssh/id_dsa -C "your comment"
```

This generates the public and private key in /private/var/root/.ssh: id_dsa and id_dsa.pub. Specifying a machine-specific comment at this step will be helpful for the management of authorized keys that you'll copy to your remote host. Replace "your comment" with something useful, like the IP address or hostname of the machine on which the key was created.

The public key is the "fingerprint" of the private key. Whenever you try to authenticate to another host, the remote machine will use your public key (which you need to copy over to the remote machine first, I'll describe that in a bit) to verify the temporary key that ssh generates on the fly based on your private key. The public key can be used to verify that the temporary key could only have been generated by the same private key that generated the public key. If you entered a passphrase when you created your keys, you will be required to enter this passphrase when you try to authenticate (which is why using passphrases won't work with an unattended script). The beauty of using public key authentication is that it is more secure because the encryption is more difficult to break than a password and you never transfer a password over the network.

Next, the id_dsa.pub (the public authentication key) needs to be added to the "/var/root/.ssh/authorized_keys" file on the machine you intend to back up to (e.g. the "server").

Create or update the authorized_keys files

Now we need to copy the client's public key to the server's authorized_keys file. We can do this several ways, but the easiest is with a simple Terminal command that appends the public key to the list of authorized keys:

```
sudo cat /private/var/root/.ssh/id_dsa.pub | ssh root@remote_address 'cat -
>> ~/.ssh/authorized_keys'
```

If you do this multiple times, you'll get multiple copies of your public key in the server's authorized_keys file. While this is harmless, it isn't a good practice, so consider editing that file manually to prune out old keys.

(Optionally) Limit the remote client's access to rsync only

If you want to limit use of the public key authentication method to only certain commands, you can implement a wrapper script that ssh will execute prior to running the requested

command. I use **this script** to limit ssh command requests to only rsync.

To implement this wrapper script, we will copy the wrapper script to /private/etc on the server, then we will modify the authorized_keys file of the user that we are using to authenticate on the server.

```
cd ~/.ssh
pico authorized_keys
```

Position the cursor at the beginning of the file and enter:

```
command="/private/etc/rsync-wrapper.sh"
```

Save and close the file.

Run rsync manually to test your settings

A complete description of all the options to rsync 3.0.6 is outside of the scope of this article. The following is a command that will perform an incremental backup of your root filesystem to a backup device on your server:

```
/usr/local/bin/rsync -aNHAXx --protect-args --fileflags --force-change --
rsync-path="/usr/local/bin/rsync" /
root@remote_address:/Volumes/Backup/MyClient
```

Create a LaunchDaemon to automate rsync

A LaunchDaemon can be used to automate using rsync for unattended backups, synchronization, or updates. Here is an **article on MacResearch.org on that topic**. Note that **Carbon Copy Cloner version 3** also leverages rsync and LaunchDaemons for unattended backups.

Rsync backup fidelity

I'm happy to report that rsync 3.0.6 passes the **backup bouncer test suite** with flying colors. Here are the results of my testing (using **`aNHAX --fileflags --force-change`**):

```
bash-3.2# ./bbouncer verify -d /Volumes/CCC_Source/ /Volumes/CCC_Target/
Verifying:    basic-permissions ... ok
Verifying:           timestamps ...
   Sub-test:    modification time ... ok
ok
Verifying:             symlinks ... ok
Verifying:    symlink-ownership ... ok
Verifying:            hardlinks ... ok
Verifying:       resource-forks ... ok
Verifying:         finder-flags ... ok
Verifying:         finder-locks ... ok
Verifying:        creation-date ... ok
Verifying:            bsd-flags ... ok
Verifying:       extended-attrs ...
   Sub-test:             on files ... ok
   Sub-test:       on directories ... ok
   Sub-test:          on symlinks ... ok
ok
Verifying: access-control-lists ...
   Sub-test:             on files ... ok
   Sub-test:              on dirs ... ok
ok
Verifying:               fifo ... ok
Verifying:             devices ... ok
Verifying:          combo-tests ...
   Sub-test:  xattrs + rsrc forks ... ok
   Sub-test:     lots of metadata ... ok
ok
```