By icarus

Table of Contents

The Need For Speed

A few years ago, as part of a contract we were servicing, I was asked to take over Webmaster responsibilities for a suite of open−source Web applications we were developing. An important component of the project was its open, or public, nature − users were invited to participate directly in the development process by providing feedback on development snapshots released by the development team to the Web site on a daily basis. Since our customers for the software were located in a different country, these daily development snapshots also provided them with an easy way to check on the progress of the project at any time.

As Webmaster for the project Web site, it became my responsibility to ensure that the site was always running the latest build of the software, so that users could play with it and give the development team feedback on how well it was (or wasn't) working. In the beginning, the task was easy − but as the project size grew, I found myself spending more and more time in front of my FTP client, watching as one file after another slowly wended its way from our staging server to our Web host.

Now, you have to keep in mind that, at this time, no one had heard of broadband, and so most of these uploads took place over a slow modem link. Since I had no way of knowing which files had changed between builds, I usually just uploaded the entire source tree from our local servers to the Web server − a long process, and one which grew ever longer as the project matured and the development team added new features. What I *really* needed was a way to just transfer the delta − the changes between the last build and the current one − so as to reduce both the time spent by me on the task, and the cost to the company in terms of connectivity charges.

That was when I discovered rsync.

Getting The Skinny

Rsync, in the words of its official Web site at http://www.samba.org/rsync/, is a "faster, flexible replacement for rcp". (for those of you not clued into the lingo, rcp is a remote shell program which allows you to copy files from one host to another). Like rcp, rsync allows you to transfer files between hosts; however, unlike rcp, rsync attempts to identify differences between source and destination files prior to initiating a transfer, and (assuming differences exist) tries only to copy the changes, rather than the entire file.

Needless to say, this is far more complicated than it seems − rsync accomplishes it via a specially−designed algorithm that allows it to obtain a list of all the differences between the source and destination files, and transfer these differences only. You don't need to worry about the details − if you're really interested, you can read about rsync's internals at http://dp.samba.org/rsync/tech_report/ − but you certainly should be impressed with the end result: a substantial reduction in both bandwidth and time used, all accomplished by the simple expedient of transferring only the differences between files, rather than the entire file.

In addition to the differential−search algorithm that makes up the core of rsync, the program also comes with a bunch of other useful features. Files (and not just files, oh no − entire directories, devices and links too!) can be copied from one host to another with permissions and other file attributes intact. Support for two−way transfer substantially simplifies the task of mirroring data between hosts. Built−in authentication makes it simple to protect access to sensitive files, and data can be transmitted over SSH for greater security.

Intrigued? Wanna see it in action? Flip the page, and let's get installing!

Building Blocks

The first order of business to install rsync on the your Linux box. Drop by the official rsync Web site at http://www.samba.org/rsync/ and get yourself the latest stable release of the software (this tutorial uses rsync 2.5.5).

Once you've downloaded the source code archive to your Linux box (mine is named "olympus"), log in as "root"

```
[me@olympus] $ su -
Password: ****
```

and extract the source to a temporary directory.

```
[root@olympus] $ cd /tmp
[root@olympus] $ tar -xzvf /home/me/rsync-2.5.5.tar-gz
```

Next, configure the package using the provided "configure" script,

```
[root@olympus] $ cd /tmp/rsync-2.5.5
[root@olympus] $ ./configure
```

and compile and install it.

```
[root@olympus] $ make
[root@olympus] $ make install
```

Unless you specified a different path to the "configure" script, rsync will have been installed to the directory "/usr/local/bin".

rsync can be used to compare files on the same physical machine, or between two different hosts on your network. If you're planning to sync files between two hosts, you should make sure that rsync is installed on both hosts − simply follow the process above to install the program on each host.

Once you've got rsync installed, the next step is to take it for a test drive.

Temporary Insanity

The first (and most basic) thing you can do with rsync involves using it as a more intelligent copy command on your local system. So, let's suppose that I have the following directory tree in my home area on my Linux box,

```
[me@olympus] $ tree -d /home/me
|-- Desktop
| |-- Autostart
| |-- Templates
| `-- Trash
|-- bin
|-- mail
|-- public_html
| |-- cache
| `-- chatserver
| |-- chat
| | |-- admin
| | |-- config
| | |-- images
| | | |-- admin
| | | |-- smilies
| | | `-- tutorials
| | |-- install
| | | |-- database
| | | `-- languages
| | |-- lib
| | | |-- commands
| | | `-- database
| | `-- localization
| | `-- english
| `-- docs
|-- test
`-- tmp

27 directories
```

and let's further suppose that I need to copy my personal Web pages to a backup folder ("/mnt/zip/backup") on the same system.

```
[me@olympus] $ ls -l /mnt/zip/backup
total 0
```

With rsync, accomplishing this is a snap:

```
[me@olympus] $ rsync --verbose --stats --recursive public_html
/mnt/zip/backup/

Number of files: 177
Number of files transferred: 158
Total file size: 1043209 bytes
Total transferred file size: 1043209 bytes
Literal data: 0 bytes
Matched data: 1043209 bytes
File list size: 3453
Total bytes written: 15469
Total bytes read: 12018
```

As with a regular "cp" command, rsync needs to know the source and destination for the copy operation. Once it has this information, it compares the files in the two locations and updates the destination to an exact replica of the source. Let's see if it worked as advertised:

```
[me@olympus] $ ls /mnt/zip/backup/
public_html
```

Yup, it did − as you can see, my backup folder now contains a copy of my Web pages.

You can specify more than one source directory to be copied as well − let me add my "mail" directory to the list and run the command again.

```
[me@olympus] $ rsync --verbose --stats --recursive public_html
mail
/mnt/zip/backup/

Number of files: 181
Number of files transferred: 161
Total file size: 1043209 bytes
Total transferred file size: 1043209 bytes
Literal data: 0 bytes
Matched data: 1043209 bytes
File list size: 3513
Total bytes written: 15637
Total bytes read: 12066

wrote 15637 bytes read 12066 bytes 18468.67 bytes/sec
total size is 1043209 speedup is 37.66
```

Next, let's make a few changes to the original files, and see if rsync can detect them and selectively update the destination the next time I sync up.

```
[me@olympus] $ touch public_html/a.dat
[me@olympus] $ ls > public_html/a.dat
[me@olympus] $ touch public_html/b.dat
[me@olympus] $ rm public_html/cache/test.html
[me@olympus] $ vi public_html/error.php
```

As you can see, I've added a couple of new files, deleted one old file and made changes to one PHP script. Let's sync up again and see what happens.

```
[me@olympus] $ rsync --verbose --stats --recursive public_html
/mnt/zip/backup/
```

Pretty cool, huh? rsync added the two extra files to my backup, and identified and copied the modified file as well. However, the single file I deleted from the source is still present in the backup − obviously, rsync didn't delete it.

A quick look at the rsync documentation clears that one up − by default, rsync doesn't delete files from the destination when synchronizing directories. This default behaviour can be overridden by adding the "−−delete" parameter to the rsync command line.

```
[me@olympus] $ rsync --verbose --stats --recursive --delete
public_html
/mnt/zip/backup/
```

And now my destination is an exact copy of my source.

```
[me@olympus] $ ls /mnt/zip/backup/public_html/cache/

index.php m2_h.gif m2_n.gif tmp.gif
```

It should be noted that the "−−delete" option can cause substantial damage if used unwisely − the rsync manual suggests always performing a dry run first when using this option.

Remote Control

So that takes care of local sync − now how about remote sync?

In order to synchronize files between two hosts, it's necessary to run rsync in daemon (server) mode on one of the hosts. When running in this mode, rsync exposes a list of directories on the server; any remote host running rsync can then connect to this server and copy files to or from it.

Before you can run rsync in daemon mode, you need to configure it. This is accomplished via a configuration file, usually "/etc/rsyncd.conf" (although you can specify a different file as well, via the "−−config" command−line argument). Here's an example:

```
log file = /var/log/rsyncd.log

[home]
path = /home/me
comment = My Home Area
list = yes
read only = no
```

As you can see, this configuration file is similar to a standard Windows INI file, in that it is broken up into different sections or "modules", each containing variable−value pairs. Modules are identified by square braces around the module name, and lines beginning with semi−colons (;) or hashes (#) are treated as comments and ignored.

The first part of the file sets up global variables for rsync to use − in this case, it specifies the log file for rsync to use. It's also possible to specify, in this section, a welcome message that is displayed when a client attempts to connect to the server.

```
log file = /var/log/rsyncd.log
motd file = /var/log/message.txt
```

The second part sets up a module on the server − this is simply a directory that is available to all connecting clients. In this case, I've selected the "/home/me" directory, given it the share name "home" and set it to be writeable by all users.

```
[home]
path = /home/me
comment = My Home Area
list = yes
read only = no
```

The

```
path = /home/me
```

option tells rsync where to locate the module on the server, while the

```
list = yes
```

option tells it to include the module in the list returned to connecting clients.

By default, modules on the server are not writable − that is, clients cannot upload files to the corresponding directories. This default behaviour cane be corrected via the extra

```
read only = no
```

option.

Once the configuration file has been saved, it's time to start up rsync in daemon mode.

```
[me@olympus] $ rsync --daemon
```

If the rsync daemon starts up OK, you can attempt to connect to it from another host. Let's assume that this second host is named "xanadu", and already has rsync installed on it. What I'd like to do is transfer my home directory on "olympus" − the same one I backed up on the previous page − to "xanadu". Here's how I'd go about it:

```
[me@xanadu] $ rsync --verbose --progress --stats --recursive
olympus::home/ .

Number of files: 230
Number of files transferred: 187
Total file size: 1054649 bytes
Total transferred file size: 1054649 bytes
Literal data: 1054649 bytes
Matched data: 0 bytes
File list size: 4318
Total bytes written: 3052
Total bytes read: 1066527

wrote 3052 bytes read 1066527 bytes 713052.67 bytes/sec
```

```
total size is 1054649 speedup is 0.99

[me@xanadu] $ ls
bin Desktop mail public_html test tmp
```

Synchronization need not be in one direction only. Using exactly the same setup as above − an rsync server on "olympus" and an rsync client on "xanadu" − it's also possible for me to transfer files in the other direction. Consider the following example, which illustrates by copying the "sql" directory to my home area on "olympus":

```
[me@xanadu] $ rsync --verbose --progress --stats --recursive
sql
olympus::home/
```

Doing More

It's possible to obtain a list of all the modules available on the rsync server by omitting the module name from the command line when connecting to the server. Here's an example, and the output:

```
[me@xanadu] $ rsync olympus::
home My Home Area
```

Now, if I were to add a few more modules to the configuration file,

```
[temp]
path = /tmp
comment = Temp Area
list = yes
```

restart the rsync server on "olympus",

```
[me@olympus] $ killall rsync
[me@olympus] $ rsync --daemon
```

and attempt to reconnect to it from "xanadu", I'd have access to the new modules as well.

```
[me@xanadu] $ rsync olympus::
home My Home Area
temp Temp Area
```

You can exclude modules from being listed in this manner by specifying a

```
list = no
```

option within the module configuration.

It's also possible to tell rsync to exclude certain files from the synchronization process, with the "--exclude" command-line option. Here's an example, which copies all the files *except* those with a ".tmp" extension from "xanadu" to "olympus":

```
[me@xanadu] $ rsync --verbose --progress --stats --recursive
--exclude="*.tmp" olympus::home/ .
```

Finally, rsync's default behaviour when encountering symbolic links is to omit them − as in the following example:

```
[me@xanadu] $ rsync --progress --recursive olympus::home/ .
skipping non-regular file "public_html/config.lib.php3"
skipping
non-regular file "public_html/start.php"
```

As you can see, when I attempt to copy the "/home/me/public_html" directory to "xanadu", every symbolic link within that directory is skipped. You can have rsync retain these links as is during the copy process by specifying the "−−links" option on the command line,

```
[me@xanadu] $ rsync --progress --recursive --links
olympus::home/ .
```

or replace the symbolic links by the actual files being referenced with the "−−copy−links" option.

```
[me@xanadu] $ rsync --progress --recursive --copy-links
olympus::home/
.
```

What's In A Name?

Thus far, all the examples you've seen have involved so−called anonymous access to the rsync server − any user could connect to the host server and transfer files between the two systems. Needless to say, this is both insecure and dangerous − it's quite possible, for example, for someone to mistakenly sync up an empty directory with the "−−delete" option, thereby destroying files on the destination machine.

In order to add a greater level of security, therefore, rsync comes with a simple authentication scheme, which requires users to log in to the rsync server with a password before performing any file transfer operation. This authentication can be activated on a per−module basis, and involves adding the "auth users" and "secrets file" variables to each module in the configuration file.

The "auth users" variable tells rsync which users are authorized to access the corresponding module on the server, while the "secrets file" variable tells rsync which file to use for password authentication. Here's an example:

```
[home]
path = /home/me
auth users = john, joe, sherry
secrets file = /home/me/rsync−users
```

In this case, only the users "john", "joe" and "sherry" are permitted access to the module "home", and their passwords can be verified against the data in the file "rsync−users". It's important to ensure that this file is not world−readable.

This secrets file is a simple text file containing a list of comma−separated usernames and passwords, each set on a new line. Here's an example:

```
[me@olympus] $ cat rsync−users
john:john
joe:joe
sherry:g5473m
```

Note that these users need not necessarily be "real" users on the system.

Let's now update our configuration file to include some authentication for the "home" module, and restart the rsync daemon on "olympus":

```
[home]
path = /home/me
comment = My Home Area
list = yes
read only = no
```

```
auth users = john, joe
secrets file = /tmp/rsync-users
```

This time, when I attempt to connect to the rsync server from "xanadu", look what happens:

```
[me@xanadu] $ rsync --progress --recursive joe@olympus::home/
.
Password: ***
```

It's only after entering the correct password for user "joe" that I'm allowed access to the module. Note the manner in which the username is specified, by prefixing it to the host name on the command line.

Finally, you can use SSH for your rsync transfers by specifying the path to the "ssh" binary in your rsync command line:

```
[me@xanadu] $ rsync -rsh=/usr/bin/ssh --progress --recursive
olympus:home/ .
```

In this case, rsync will use SSH to perform the transaction. Note the single colon in the destination host name − this tells rsync to use the SSH shell instead of connecting to the rsync server directly.

You can also use the "hosts allow" and "hosts deny" options to restrict access to the server by host − take a look at the documentation for details.

Mirror, Mirror, On The Wall

So that's the theory. Let's now see how I applied it to my original problem (in case you've forgotten, I needed to copy the contents of a directory on our staging server to a corresponding directory on our Web server).

Let's assume that the staging server is called "medusa", and the directory to be mirrored from it on to the live server is "/usr/local/apache/htdocs/beta". The first thing to do, obviously, was to set up rsync as a daemon on one of the hosts − say "medusa" − and configure it to make the "/usr/local/apache/htdocs/beta" directory available as a module.

```
[web]
path = /usr/local/apache/htdocs
comment = Web Server Root
list = yes
read only = yes
```

Next, I needed to log in to the other end of the connection − the live Web server − and run rsync to connect to the staging server and get the latest build released by the development team to "medusa".

```
[webmaster@domain] $ cd /www-root/
[webmaster@domain] $ rsync --compress --verbose --delete
--links
--recursive --perms medusa::web/beta .
```

In case you're wondering, the "−−compress" option compresses the data while sending it, while the "−−perms" option retains the original file permissions on the destination host.

I put the two lines above into a shell script, and set it to run on a daily basis via cron. Since rsync only sends the delta when performing a copy operation, my bandwidth usage was minimal...and since the process was now largely automated, I was able to get my social life back on track.

Link Out

And that's about it. Over the course of this article, I introduced you to rsync, an extremely powerful utility for file synchronization between hosts. I showed you how to synchronize files between directories on the same machine, and also between different hosts on a network. I also demonstrated the process of configuring an rsync server, with examples of both anonymous and authenticated access, and showed you how you could perform your file transfer within an SSH connection for greater security. Finally, I illustrated how all this theory could be put to practical use with examples of how rsync could be used in common real−world situations involving file backup and mirroring.

If you'd like to read more about rsync, you should consider bookmarking the following links:

The official rsync Website, at http://www.samba.org/rsync/

rsync documentation, at http://www.samba.org/rsync/documentation.html

Usage examples, at http://www.samba.org/rsync/examples.html, and more rsync resources, at http://www.samba.org/rsync/resources.html

The SSH Web site, at http://www.ssh.org/

Till next time...stay healthy!

Note: All examples in this article have been tested on Linux/i586 with PHP 4.2.3. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!