rsnapshot HOWTO

Nathan Rosenquist <nathan@rsnapshot.org>

2004-01-20

Revision History

Revision 1.0.0	2005-01-31	NR
	Updated for rsnapshot 1.2.0	
Revision 0.9.7	2005-01-17	NR
	Spelling corrections submitted by Nicolas Kaiser	
Revision 0.9.6	2004-12-13	NR
	Misc. updates	
Revision 0.9.5	2004-07-10	NR
	Relicensed document under GPL, instead of FDL	
Revision 0.9.4	2004-07-02	NR
	Added description of proper crontab time settings	
Revision 0.9.3	2004-06-11	NR
	Misc. updates	
Revision 0.9.2	2004-05-16	NR
	Updated --link-dest info	
Revision 0.9.1	2004-01-20	NR
	Added --link-dest info	
Revision 0.9	2004-01-10	NR
	First draft	

rsnapshot is a filesystem backup utility based on rsync. Using rsnapshot, it is possible to take snapshots of your filesystems at different points in time. Using hard links, rsnapshot creates the illusion of multiple full backups, while only taking up the space of one full backup plus differences. When coupled with ssh, it is possible to take snapshots of remote filesystems as well. This document is a tutorial in the installation and configuration of rsnapshot.

Table of Contents

1. Introduction

rsnapshot is a filesystem backup utility based on rsync. Using rsnapshot, it is possible to take snapshots of your filesystems at different points in time. Using hard links, rsnapshot creates the illusion of multiple full backups, while only taking up the space of one full backup plus differences. When coupled with ssh, it is possible to take snapshots of remote filesystems as well.

rsnapshot is written in Perl, and depends on rsync. OpenSSH, GNU cp, GNU du, and the BSD logger program are also recommended, but not required. All of these should be present on most Linux systems. rsnapshot is written with the lowest common denominator in mind. It only requires at minimum Perl 5.004 and rsync. As a result of this, it works on pretty much any UNIX-like system you care to throw at it. It has been successfully tested with Perl 5.004 through 5.8.2, on Debian, Redhat, Fedora, Solaris, Mac OS X, FreeBSD, OpenBSD, NetBSD, and IRIX.

The latest version of the program and this document can always be found at http://www.rsnapshot.org/.

1.1. What you will need

At a minimum: *perl, rsync*

Optionally: *ssh, logger, GNU cp, GNU du*

Additionally, it will help if you have reasonably good sysadmin skills.

1.2. Copyright and License

1.3. Disclaimer

1.4. Feedback

Feedback is most certainly welcome for this document. Send your additions, comments and criticisms to the following email address : <nathan@rsnapshot.org>.

2. Motivation

I originally used Mike Rubel's shell scripts to do rsync snapshots a while back. These worked very well, but there were a number of things that I wanted to improve upon. I had to write two shell scripts that were customized for my server. If I wanted to change the number of intervals stored, or the parts of the filesystem that were archived, that meant manually editing these shell scripts. If I wanted to install them on a different server with a different configuration, this meant manually editing the scripts for the new server, and hoping the logic and the sequence of operations was correct. Also, I was doing all the backups locally, on a single machine, on a single hard drive (just to protect from dumb user mistakes like deleting files). Never the less, I continued on with this system for a while, and it did work very well.

Several months later, the IDE controller on my web server failed horribly (when I typed **/sbin/shutdown**, it said the command was not found). I was then faced with what was in the back of my mind all along: I had not been making regular remote backups of my server, and the local backups were of no use to me since the entire drive was corrupted. The reason I had only been making sporadic, partial remote backups is that they weren't automatic and effortless. Of course, this was no one's fault but my own, but I got frustrated enough to write a tool that would make automated remote snapshots so easy that I wouldn't ever have to worry about them again. This goal has long been reached, but work on rsnapshot still continues as people submit patches, request features, and ways are found to improve the program.

3. Installation

This section will walk you through the installation of rsnapshot, step by step. This is not the only way to do it, but it is a way that works and that is well documented. Feel free to improvise if you know what you're doing.

This guide assumes you are installing rsnapshot 1.2.0 for the first time. If you are upgrading from an earlier version, please read the `INSTALL` file that comes with the source distribution instead.

3.1. 30 second version (for the impatient)

```
./configure --sysconfdir=/etc
su
make install
cp /etc/rsnapshot.conf.default /etc/rsnapshot.conf
```

The rest of this section is the long version.

3.2. Untar the source code package

```
tar xzvf rsnapshot-1.2.0.tar.gz
```

If you don't have GNU `tar`, you may have to do this in two steps instead:

```
gunzip rsnapshot-1.2.0.tar.gz
tar xvf rsnapshot-1.2.0.tar
```

3.3. Change to the source directory

```
cd rsnapshot-1.2.0/
```

3.4. Decide where you want to install

By default, the installation procedure will install all files under `/usr/local`. For this tutorial, this will be OK except we will install the config file under `/etc`.

We are assuming that `rsync`, `ssh`, `logger`, and `du` are all in your search path. If this is not the case, you can specify the path to any of these programs using the typical Autoconf *`--with-program=/path/to/program`* syntax. For example, if Perl was in `/opt/bin/perl` and rsync was in `/home/me/bin/rsync`, you could run configure like:

```
./configure --with-perl=/opt/bin/perl --with-rsync=/home/me/bin/rsync
```

3.5. Run the configure script

This will poke and prod your system to figure out where the various external programs that rsnapshot depends on live. It also generates the Makefile that we will use to install the program. The configure script accepts arguments that can be used to tell it where to install the program, and also where to find the supporting programs. For this installation, the only non-default option we want is to put the config file in the `/etc` directory. To do this, run this command at the shell:

```
./configure --sysconfdir=/etc
```

If all goes well, you're ready to install the program. If there was a problem, it should be descriptive. Most likely a problem would be the result of something that was required and not found (like rsync or perl). If this happens, you must figure out where the missing program is located on your system, or install it if necessary. If you know where it is but configure couldn't find it, you can specify the path using the *`--with-program=/path/to/program`* options described above.

3.6. Install the program

If you've followed these instructions so far, you will have configured rsnapshot to be installed under `/usr/local`, with the config file in `/etc`. Under these circumstances, it will be necessary to become root to install the program. Now is the time to do so. You will, of course, need the root password to do this:

```
su
```

This will prompt you for the root password.

Now, to install rsnapshot, run the following command:

```
make install
```

This will install rsnapshot with all the settings you specified in the ./configure stage. If all goes well, you will have the following files on your system:

`/usr/local/bin/rsnapshot` The rsnapshot program

`/usr/local/man/man1/rsnapshot.1` Man page

`/etc/rsnapshot.conf.default` The example config file

If you decide later that you don't want rsnapshot on your system anymore, simply remove the files listed above, or run **make uninstall** in the same source directory you installed from. Of course, if you installed

with different options, the location of these files may be different.

4. Configuration

4.1. Create the config file

In the install process, the config file is not created or installed. However, a working example is provided that you can copy. To copy the example config file into the location rsnapshot will be looking for the real config file:

```
cp /etc/rsnapshot.conf.default /etc/rsnapshot.conf
```

As a general rule, you should avoid modifying `/etc/rsnapshot.conf.default`, simply because it is a working example that you may wish to refer to later. Also, if you perform an upgrade, the `rsnapshot.conf.default` file will always be upgraded to the latest version, while your real config file will be safe out of harm's way. Please note that if you run **make upgrade** during an upgrade, your rsnapshot.conf may be modified slightly, and the original will then be saved in `rsnapshot.conf.backup` in the same directory.

4.2. Where to go for more info

The `rsnapshot.conf` config file is well commented, and much of it should be fairly self-explanatory. For a full reference of all the various options, please consult the rsnapshot man page. Type:

```
man rsnapshot
```

This will give you the complete documentation. However, it assumes that you already know what you're doing to a certain extent. If you just want to get something up and running, this tutorial is a better place to start. If your system can't find the man page, `/usr/local/man` probably isn't in your $MANPATH environmental variable. This is beyond the scope of this document, but if it isn't working for you, you can always read the newest man page on the rsnapshot web site at http://www.rsnapshot.org/

4.3. Modifying the config file

In this example, we will be using the `/.snapshots/` directory to hold the filesystem snapshots. This is referred to as the “snapshot root”. Feel free to put this anywhere you have lots of free disk space. However, the examples in this document assume you have not changed this parameter, so you will have to substitute this in your commands if you put it somewhere else.

Also please note that fields are separated by tabs, not spaces. The reason for this is so it's easier to specify file paths with spaces in them.

4.3.1. cmd_cp

If enabled, the *cmd_cp* parameter should contain the path to the GNU `cp` program on your filesystem. If you are using Linux, be sure to uncomment this by removing the hash mark (#) in front of it. If you are using BSD, Solaris, IRIX, or most other UNIX variants, you should leave this commented out.

What makes GNU `cp` so special is that unlike the traditional UNIX `cp`, it has the ability to make recursive “copies” of directories as hard links.

If you don't have GNU `cp`, there is a subroutine in rsnapshot that somewhat approximates this functionality (although it won't support more esoteric files such as device nodes, FIFOs, sockets, etc). This gets followed up by another call to rsync, which transfers the remaining special files, if any. In this way,

rsnapshot can support all file types on every platform.

The rule of thumb is that if you're on a Linux system, leave *cmd_cp* enabled. If you aren't on a Linux system, leave *cmd_cp* disabled. There are reports of GNU `cp` working on BSD and other non-Linux platforms, but there have also been some cases where problems have been encountered. If you enable *cmd_cp* on a non-Linux platform, please let the mailing list know how it worked out for you.

4.3.2. cmd_rsync

The *cmd_rsync* parameter must not be commented out, and it must point to a working version of `rsync`. If it doesn't, the program just will not work at all.

Please note that if you are using IRIX, there is another program named `rsync` that is different than the "real" `rsync` most people know of. If you're on an IRIX machine, you should double check this.

4.3.3. cmd_ssh

If you have `ssh` installed on your system, you will want to uncomment the *cmd_ssh* parameter. By enabling `ssh`, you can take snapshots of any number of remote systems. If you don't have `ssh`, or plan to only take snapshots of the local filesystem, you may safely leave this commented out.

4.3.4. cmd_logger

The *cmd_logger* parameter specifies the path to the `logger` program. `logger` is a command line interface to syslog. See the `logger` man page for more details. `logger` should be a standard part of most UNIX-like systems. It appears to have remained unchanged since about 1993, which is good for cross-platform stability. If you comment out this parameter, it will disable syslog support in rsnapshot. It is recommended that you leave this enabled.

4.3.5. cmd_du

The *cmd_du* parameter specifies the path to the `du` program. `du` is a command line tool that reports on disk usage. rsnapshot uses `du` to generate reports about the actual amount of disk space taken up, which is otherwise difficult to estimate because of all the hard links.

If you comment this out, rsnapshot will try to use the version of `du` it finds in your path, if possible. The GNU version of `du` is recommended, since it has the best selection of features, and supports the most options. The BSD version also seems to work, although it doesn't support the **-h** flag. Solaris **du** does not work at all, because it doesn't support the **-c** parameter.

4.3.6. link_dest

If you have `rsync` version 2.5.7 or later, you may want to enable this. With *link_dest* enabled, rsnapshot relies on rsync to create recursive hard links, overriding GNU cp in most, but not all, cases. With *link_dest* enabled, every single file on your system can be backed up in one pass, on any operating system. To get the most out of rsnapshot on non-Linux platforms, *link_dest* should be enabled. Be advised, however, that if a remote host is unavailable during a backup, rsnapshot will take an extra step and roll back the files from the previous backup. Using GNU cp, this would not be necessary.

4.3.7. interval

rsnapshot has no idea how often you want to take snapshots. Everyone's backup scheme may be different. In order to specify how much data to save, you need to tell rsnapshot which "intervals" to keep, and how many of each. An interval, in the context of the rsnapshot config file, is a unit of time measurement. These can actually be named anything (as long as it's alphanumeric, and not a reserved word), but by convention we will call ours *hourly* and *daily*. In this example, we want to take a snapshot every four hours, or six times a day (these are the *hourly* intervals). We also want to keep a second set, which are

taken once a day, and stored for a week (or seven days). This happens to be the default, so as you can see the config file reads:

```
interval    hourly  6
interval    daily   7
```

It also has some other entries, but you can either ignore them or comment them out for now.

Please note that the *hourly* interval is specified first. This is very important. The first *interval* line is assumed to be the smallest unit of time, with each additional line getting successively larger. Thus, if you add a *yearly* interval, it should go at the bottom, and if you add a *minutes* interval, it should go before hourly. It's also worth noting that the snapshots get “pulled up” from the smallest interval to the largest. In this example, the daily snapshots get pulled from the oldest hourly snapshot, not directly from the main filesystem.

4.3.8. backup

Please note that the destination paths specified here are based on the assumption that the *--relative* flag is being passed to `rsync` via the *rsync_long_args* parameter. If you are installing for the first time, this is the default setting. If you upgraded from a previous version, please read the `INSTALL` file that came with the source distribution for more information.

This is the section where you tell rsnapshot what files you actually want to back up. You put a “backup” parameter first, followed by the full path to the directory or network path you're backing up. The third column is the relative path you want to back up to inside the snapshot root. Let's look at an example:

```
backup      /etc/      localhost/
```

In this example, *backup* tells us it's a backup point. `/etc/` is the full path to the directory we want to take snapshots of, and `localhost/` is a directory inside the *snapshot_root* we're going to put them in. Using the word *localhost* as the destination directory is just a convention. You might also choose to use the server's fully qualified domain name instead of *localhost*. If you are taking snapshots of several machines on one dedicated backup server, it's a good idea to use their various hostnames as directories to keep track of which files came from which server.

In addition to full paths on the local filesystem, you can also backup remote systems using `rsync` over `ssh`. If you have `ssh` installed and enabled (via the *cmd_ssh* parameter), you can specify a path like:

```
backup      root@example.com:/etc/     example.com/
```

This behaves fundamentally the same way, but you must take a few extra things into account.

- The ssh daemon must be running on example.com
- You must have access to the account you specify the remote machine, in this case the root user on example.com.
- You must have key-based logins enabled for the root user at example.com, without passphrases. If you wanted to perform backups as another user, you could specify the other user instead of root for the source (i.e. user@domain.com). Please note that allowing remote logins with no passphrase is a security risk that may or may not be acceptable in your situation. Make sure you guard access to the backup server very carefully! For more information on how to set this up, please consult the ssh man page, or a tutorial on using ssh public and private keys. You will find that the key based logins are better in many ways, not just for rsnapshot but for convenience and security in general. One thing you can do to mitigate the potential damage from a backup server breach is to create alternate users on the client machines with uid and gid set to 0, but with a more restrictive shell such as scponly.

- This backup occurs over the network, so it may be slower. Since this uses `rsync`, this is most noticeable during the first backup. Depending on how much your data changes, subsequent backups should go much, much faster since *rsync* only sends the differences between files.

4.3.9. backup_script

With this parameter, the second column is the full path to an executable backup script, and the third column is the local path you want to store it in (just like with the "backup" parameter). For example:

```
backup_script	/usr/local/bin/backup_pgsql.sh	localhost/postgres/
```

In this example, rsnapshot will run the script `/usr/local/bin/backup_pgsql.sh` in a temp directory, then sync the results into the `localhost/postgres/` directory under the snapshot root. You can find the backup_pgsql.sh example script in the `utils/` directory of the source distribution. Feel free to modify it for your system.

Your backup script simply needs to dump out the contents of whatever it does into it's current working directory. It can create as many files and/or directories as necessary, but it should not put its files in any pre-determined path. The reason for this is that rsnapshot creates a temp directory, changes to that directory, runs the backup script, and then syncs the contents of the temp directory to the local path you specified in the third column. A typical backup script would be one that archives the contents of a database. It might look like this:

```
#!/bin/sh

/usr/bin/mysqldump -uroot mydatabase > mydatabase.sql
/bin/chmod 644 mydatabase.sql
```

There are several example scripts in the `utils/` directory of the rsnapshot source distribution to give you more ideas.

Make sure the destination path you specify is unique. The backup script will completely overwrite anything in the destination path, so if you tried to specify the same destination twice, you would be left with only the files from the last script. Fortunately, rsnapshot will try to prevent you from doing this when it reads the config file.

Please remember that these backup scripts will be invoked as the user running rsnapshot. In our example, this is root. Make sure your backup scripts are owned by root, and not writable by anyone else. If you fail to do this, anyone with write access to these backup scripts will be able to put commands in them that will be run as the root user. If they are malicious, they could take over your server.

4.4. Testing your config file

When you have made all your changes, you should verify that the config file is syntactically valid, and that all the supporting programs are where you think they are. To do this, run rsnapshot with the configtest argument:

```
rsnapshot configtest
```

If all is well, it should say `Syntax OK`. If there's a problem, it should tell you exactly what it is. Make sure your config file is using tabs and not spaces, etc.

The final step to test your configuration is to run rsnapshot in test mode. This will print out a verbose list of the things it will do, without actually doing them. To do a test run, run this command:

```
rsnapshot -t hourly
```

This tells rsnapshot to simulate an "hourly" backup. It should print out the commands it will perform when it runs for real. Please note that the test output might be slightly different than the real execution, but only because the test doesn't actually do things that may be checked for later in the program. For example, if the program will create a directory and then later test to see if that directory exists, the test run might claim that it would create the directory twice, since it didn't actually get created during the test. This should be the only type of difference you will see while running a test.

5. Automation

Now that you have your config file set up, it's time to set up rsnapshot to be run from cron. As the root user, edit root's crontab by typing:

```
crontab -e
```

You could alternately keep a crontab file that you load in, but the concepts are the same. You want to enter the following information into root's crontab:

```
0 */4 * * *       /usr/local/bin/rsnapshot hourly
30 23 * * *       /usr/local/bin/rsnapshot daily
```

It is usually a good idea to schedule the larger intervals to run a bit before the lower ones. For example, in the crontab above, notice that *daily* runs 30 minutes before *hourly*. This helps prevent race conditions where the *daily* would try to run before the *hourly* job had finished. This same strategy should be extended so that a *weekly* entry would run before the *daily* and so on.

6. How it works

We have a snapshot root under which all backups are stored. By default, this is the directory `/.snapshots/`. Within this directory, other directories are created for the various intervals that have been defined. In the beginning it will be empty, but once rsnapshot has been running for a week, it should look something like this:

```
[root@localhost]# ls -l /.snapshots/
drwxr-xr-x    7 root     root         4096 Dec 28 00:00 daily.0
drwxr-xr-x    7 root     root         4096 Dec 27 00:00 daily.1
drwxr-xr-x    7 root     root         4096 Dec 26 00:00 daily.2
drwxr-xr-x    7 root     root         4096 Dec 25 00:00 daily.3
drwxr-xr-x    7 root     root         4096 Dec 24 00:00 daily.4
drwxr-xr-x    7 root     root         4096 Dec 23 00:00 daily.5
drwxr-xr-x    7 root     root         4096 Dec 22 00:00 daily.6
drwxr-xr-x    7 root     root         4096 Dec 29 00:00 hourly.0
drwxr-xr-x    7 root     root         4096 Dec 28 20:00 hourly.1
drwxr-xr-x    7 root     root         4096 Dec 28 16:00 hourly.2
drwxr-xr-x    7 root     root         4096 Dec 28 12:00 hourly.3
drwxr-xr-x    7 root     root         4096 Dec 28 08:00 hourly.4
drwxr-xr-x    7 root     root         4096 Dec 28 04:00 hourly.5
```

Inside each of these directories is a “full” backup of that point in time. The destination directory paths you specified under the *backup* and *backup_script* parameters get stuck directly under these directories. In the example:

```
backup          /etc/           localhost/
```

The `/etc/` directory will initially get backed up into `/.snapshots/hourly.0/localhost/etc/`

Each subsequent time rsnapshot is run with the *hourly* command, it will rotate the `hourly.X` directories, and then "copy" the contents of the `hourly.0` directory (using hard links) into `hourly.1`.

When **rsnapshot daily** is run, it will rotate all the `daily.X` directories, then copy the contents of `hourly.5` into `daily.0`.

`hourly.0` will always contain the most recent snapshot, and `daily.6` will always contain a snapshot from a week ago. Unless the files change between snapshots, the "full" backups are really just multiple hard links to the same files. Thus, if your `/etc/passwd` file doesn't change in a week, `hourly.0/localhost/etc/passwd` and `daily.6/localhost/etc/passwd` will literally be the same exact file. This is how rsnapshot can be so efficient on space. If the file changes at any point, the next backup will unlink the hard link in `hourly.0`, and replace it with a brand new file. This will now take double the disk space it did before, but it is still considerably less than it would be to have full unique copies of this file 13 times over.

Remember that if you are using different intervals than the ones in this example, the first interval listed is the one that gets updates directly from the main filesystem. All subsequently listed intervals pull from the previous intervals. For example, if you had *weekly*, *monthly*, and *yearly* intervals defined (in that order), the weekly ones would get updated directly from the filesystem, the monthly ones would get updated from weekly, and the yearly ones would get updated from monthly.

7. Restoring backups

When rsnapshot is first run, it will create the *snapshot_root* directory (`/.snapshots/` by default). It assigns this directory the permissions 700, and for good reason. The snapshot root will probably contain files owned by all sorts of users on your system. If any of these files are writeable (and of course some of them will be), the users will still have write access to their files. Thus, if they can see the snapshots directly, they can modify them, and the integrity of the snapshots can not be guaranteed.

For example, if a user had write permission to the backups and accidentally ran **rm -rf /**, they would delete all their files in their home directory and all the files they owned in the backups!

7.1. root only

The simplest, but least flexible solution, is to simply deny non-root users access to the snapshot root altogether. The root user will still have access of course, and as with all other aspects of system administration, must be trusted not to go messing with things too much. However, by simply denying access to everyone, the root user will be the only one who can pull backups. This may or may not be desirable, depending on your situation. For a small setup or a single-user machine, this may be all you need.

7.2. All users

If users need to be able to pull their own backups, you will need to do a little extra work up front (but probably less work in the long run). The best way to do this seems to be creating a container directory for the snapshot root with 700 permissions, giving the snapshot root directory 755 permissions, and mounting the snapshot root for the users read-only. This can be done over NFS and Samba, to name two possibilities. Let's explore how to do this using NFS on a single machine:

Set the snapshot_root variable in `/etc/rsnapshot.conf` equal to `/.private/.snapshots/`

```
snapshot_root       /.private/.snapshots/
```

Create the container directory:

```
mkdir /.private/
```

Create the real snapshot root:

```
mkdir /.private/.snapshots/
```

Create the read-only snapshot root mount point:

```
mkdir /.snapshots/
```

Set the proper permissions on these new directories:

```
chmod 0700 /.private/
chmod 0755 /.private/.snapshots/
chmod 0755 /.snapshots/
```

In `/etc/exports`, add `/.private/.snapshots/` as a read only NFS export:

```
/.private/.snapshots/  127.0.0.1(ro,no_root_squash)
```

In `/etc/fstab`, mount `/.private/.snapshots/` read-only under `/.snapshots/`

```
localhost:/.private/.snapshots/   /.snapshots/   nfs    ro   0 0
```

You should now restart your NFS daemon.

Now mount the read-only snapshot root:

```
mount /.snapshots/
```

To test this, go into the /.snapshots/ directory as root. It is set to read-only, so even root shouldn't be able to write to it. As root, try:

```
touch /.snapshots/testfile
```

This should fail, citing insufficient permissions. This is what you want. It means that your users won't be able to mess with the snapshots either.

Now, all your users have to do to recover old files is go into the /.snapshots directory, select the interval they want, and browse through the filesystem until they find the files they are looking for. They can't modify anything in here because NFS will prevent them, but they can copy anything that they had read permission for in the first place. All the regular filesystem permissions are still at work, but the read-only NFS mount prevents any writes from happening.

Please note that some NFS configurations may prevent you from accessing files that are owned by root and set to only be readable by root. In this situation, you may wish to pull backups for root from the "real" snapshot root, and let non-privileged users pull from the read-only NFS mount.

8. Conclusion

If you followed the instructions in this document, you should now have rsnapshot installed and set up to perform automatic backups of your system. If it's not working, go back and trace your steps back to see if you can isolate the problem.

The amount of disk space taken up will be equal to one full backup, plus an additional copy of every file that is changed. There is also a slight disk space overhead with creating multiple hard links, but it's not very much. On my system, adding a second, completely identical 3 Gigabyte interval alongside the original one only added about 15 Megabytes.

You can use the *du* option to rsnapshot to generate disk usage reports. To see the sum total of all space used, try:

```
rsnapshot du
```

If you were storing backups under `localhost/home/` and wanted to see how much this subdirectory takes up throughout all your backups, try this instead:

```
rsnapshot du localhost/home/
```

The latest version of this document and the rsnapshot program can always be found at http://www.rsnapshot.org/

9. More resources

Web sites

Mike Rubel's original shell scripts, upon which this project is based	http://www.mikerubel.org/computers/rsync_snapshots/
Perl	http://www.perl.org/
GNU cp and du (coreutils package)	http://www.gnu.org/software/coreutils/
rsync	http://rsync.samba.org/
OpenSSH	http://www.openssh.org/
rsnapshot	http://www.rsnapshot.org/