wiki.archlinux.org

rsnapshot - ArchWiki

Rsnapshot is an open source utility that provides incremental back ups.

Installation

Install the rsnapshot package.

Configuration

In the install process, the config file is created. It is recommended you make a back up of this file in case you need to reconfigure the file again.

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

The `/etc/rsnapshot.conf` 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.

Root Directory

Choose the directory where you want to store the file system back ups, in this case I will store the back ups in `/mnt/backups/`

`/etc/rsnapshot.conf`

```
# All snapshots will be stored under this root directory.
#
snapshot_root   /mnt/backups/
```

Note: Fields are separated by tabs, not spaces. The reason for this is so it's easier to specify file paths with spaces in them.

External program dependencies

Uncomment the lines refering to the unix commands `cp`, `du`, `ssh` (if you want to do remote back ups) and `rsnapshot-diff`, etc. This section of the file should look like this:

`/etc/rsnapshot.conf`

```
# LINUX USERS:   Be sure to uncomment "cmd_cp". This gives you extra features.
# EVERYONE ELSE: Leave "cmd_cp" commented out for compatibility.
#
# See the README file or the man page for more details.
#
cmd_cp          /usr/bin/cp

# uncomment this to use the rm program instead of the built-in perl routine.
#
cmd_rm          /usr/bin/rm

# rsync must be enabled for anything to work. This is the only command that
# must be enabled.
#
cmd_rsync       /usr/bin/rsync

# Uncomment this to enable remote ssh backups over rsync.
#
cmd_ssh /usr/bin/ssh

# Comment this out to disable syslog support.
#
cmd_logger      /usr/bin/logger

# Uncomment this to specify the path to "du" for disk usage checks.
# If you have an older version of "du", you may also want to check the
# "du_args" parameter below.
#
cmd_du          /usr/bin/du

# Uncomment this to specify the path to rsnapshot-diff.
#
cmd_rsnapshot_diff      /usr/bin/rsnapshot-diff

# Specify the path to a script (and any optional arguments) to run right
# before rsnapshot syncs files
#
#cmd_preexec    /path/to/preexec/script

# Specify the path to a script (and any optional arguments) to run right
# after rsnapshot syncs files
#
#cmd_postexec   /path/to/postexec/script
```

Note: cmd_preexec and cmd_postexec are custom scripts to be run before and after each run of rsnapshot

Retain Previous Backups

Rsnapshot allows named backup levels that retain a given number of previous backups.

When configuring these, note that the first in the list will be the only one that actually backs up files from the file system AND rotates its own previous backups. The rest will ONLY rotate previous backups, creating its newest backup from the oldest backup created by the previous item on the list. So, the order these are listed in the config file are very important.

Replace the default "BACKUP LEVELS / INTERVALS" section in the rsnapshot config file:

`/etc/rsnapshot.conf`

```
retain  hourly  24
retain  daily   7
retain  weekly  4
retain  monthly 12
```

- When `rsnapshot hourly` is called, a new backup will be created from the file system, and saved in **<snapshot_root>/hourly.0/**. The rest of the retained backups will continue to get incremented each time the command is run. So eventually, what was **<snapshot_root>/hourly.0**, will become **<snapshot_root>/hourly.23/**. Then the next time the command is run, this will be deleted.
- When `rsnapshot daily` is called, it will create **<snapshot_root>/daily.0/** from the **<snapshot_root>/hourly.23/** backup if it exists. Otherwise, rotation works the same way.
- Likewise, when `rsnapshot weekly` is called, it will create **<snapshot_root>/weekly.0/** from the **<snapshot_root>/daily.6/** backup if it exists. The pattern follows the same for each additional retain level that is configured.

Using the above config, you could call:

`rsnapshot hourly` every hour

`rsnapshot daily` every day

`rsnapshot weekly` every week

`rsnapshot monthly` every month

This would give you a robust 12 months of backups while minimizing the space taken up by older snapshots.

If you just wanted to run this same config, but only backup daily, you would need to comment out the hourly backup level. Otherwise calling `rsnapshot daily` would never actually backup any files since it's not the first on the list.

Back up

This is the section where you tell *rsnapshot* which 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 are backing up. The third column is the relative path you want to back up to inside the snapshot root.

```
/etc/rsnapshot.conf

#############################
### BACKUP POINTS / SCRIPTS ###
#############################

# LOCALHOST
backup  /home/          localhost/
```

In this example, backup tells us it's a backup point. `/home/` is the full path to the directory we want to take snapshots of, and `localhost/` is a directory **inside** the `snapshot_root` we are 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.

Remote Systems

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:

```
/etc/rsnapshot.conf

#############################
### BACKUP POINTS / SCRIPTS ###
#############################

# example.com
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.

There is an extra `backup_script` line. 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:

```
/etc/rsnapshot.conf

backup_script      /usr/local/bin/backup_mysql.sh       localhost/mysql/
```

In this example, *rsnapshot* will run the script `/usr/local/bin/backup_mysql.sh` in a temp directory, then sync the results into the `localhost/mysql/` directory under the snapshot root.

Your backup script simply needs to dump out the contents of whatever it does into its 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:

```
backup_mysql.sh

#!/bin/sh

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

Note: 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.

Note: 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.

Testing the configuration

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 is 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.

Note: Please note that the test output might be slightly different than the real execution, but only because the test does not 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 did not actually get created during the test. This should be the only type of difference you will see while running a test.

Automation

This article or section needs language, wiki syntax or style improvements. See Help:Style for reference.

Reason: Too much code for an article. (Discuss in Talk:Rsnapshot)

Now that you have your config file set up, it's time to set up *rsnapshot* to be run automatically.

First create a service file:

```
/etc/systemd/system/rsnapshot@.service

[Unit]
Description=rsnapshot (%I) backup

[Service]
Type=oneshot
Nice=19
IOSchedulingClass=idle
ExecStart=/usr/bin/rsnapshot %I
```

Then create a timer unit for hourly:

```
/etc/systemd/system/rsnapshot-hourly.timer

[Unit]
Description=rsnapshot hourly backup

[Timer]
# Run hourly
OnCalendar=*-*-* *:00:00
Persistent=true
Unit=rsnapshot@hourly.service

[Install]
WantedBy=timers.target
```

A timer unit for daily:

```
/etc/systemd/system/rsnapshot-daily.timer

[Unit]
Description=rsnapshot daily backup

[Timer]
# 05:30 is the clock time when to start it
OnCalendar=05:30
Persistent=true
Unit=rsnapshot@daily.service

[Install]
WantedBy=timers.target
```

A timer unit for weekly:

```
/etc/systemd/system/rsnapshot-weekly.timer

[Unit]
Description=rsnapshot weekly backup

[Timer]
# Run once per week on Monday at 4:30, after daily runs
OnCalendar=Monday  *-*-* 04:30:00
Persistent=true
Unit=rsnapshot@weekly.service

[Install]
WantedBy=timers.target
```

And a timer unit for monthly:

```
/etc/systemd/system/rsnapshot-monthly.timer

[Unit]
Description=rsnapshot monthly backup

[Timer]
# Run once per month at 3:30 UTC, after daily and weekly runs
OnCalendar=*-*-1 03:30:00
Persistent=true
Unit=rsnapshot@monthly.service

[Install]
WantedBy=timers.target
```

Then finally, enable and start them:

```
# systemctl enable --now rsnapshot-hourly.timer
# systemctl enable --now rsnapshot-daily.timer
# systemctl enable --now rsnapshot-weekly.timer
# systemctl enable --now rsnapshot-monthly.timer
```

Alternatively, to manually run the service, you can simply execute:

```
# systemctl start rsnapshot@hourly # or rsnapshot@daily, rsnapshot@weekly, ...
```

Note: It is usually a good idea to schedule the larger intervals to run a bit before the lower ones. 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.

External Drives

Note: This section assumes that you have already completed the configuration described in the #Automation section above.

If the destination drive is in an external enclosure connected via USB or eSATA, it may not have mounted during boot or may otherwise be unmounted at the time *rsnapshot* is scheduled to begin. If *rsnapshot* is configured to write to a path that always exists, e.g. `/.snapshots`, the data will be backed up on whichever hard drive is mounted as the root directory rather than the desired external drive.

To remedy this situation one must configure *rsnapshot* to depend upon the disk being mounted to the expected mount point. There are two actions required: alter `/etc/fstab` and `/etc/systemd/system/rsnapshot@.service`.

Systemd will read the `/etc/fstab` file and create unit files for all of the mount points therein. For this setup we need to add one, optionally two, configuration options to the mount point. At the end of the options column for the desired mount point add `x-systemd.automount` and, if you want the mount point to unmount after inactivity, `x-systemd.idle-timeout=10m`. The value *10m* can be changed to any value you wish. See systemd.automount(5) and systemd.mount(5) for additional details about the options available.

Tip: The UUID of a partition can be found by running `blkid` as root.

An example mount point:

`/etc/fstab`

```
# /dev/sdd1
UUID=2848e78d-b05a-4477-a5f0-38f35411c269 /mnt/backups ext4
noauto,nofail,noexec,nouser,nosuid,rw,async,x-systemd.device-timeout=200ms,x-
systemd.automount,x-systemd.idle-timeout=10m 0 2
```

After changing `/etc/fstab`, run `systemctl daemon-reload` so *systemd* picks up the changes made. Check that the mount and automount units look correct:

```
# systemctl show mnt-backups.mount
Where=/.snapshots
What=/dev/sdd1
Options=rw,nosuid,noexec,relatime,x-systemd.automount
Type=ext4
TimeoutUSec=1h
…

# systemctl show mnt-backups.automount
Where=/mnt/backups
DirectoryMode=0755
…
```

Finally, edit `/etc/systemd/system/rsnapshot@.service` and add the following line in the "[Unit]" section:

`/etc/systemd/system/rsnapshot@.service`

```
[Unit]
Requires=mnt-backups.mount
After=mnt-backups.mount
```

To ensure everything is configured properly check that the rsnapshot service units now require the mount point:

```
# systemctl show rsnapshot@daily.service | grep 'Requires='
Requires=sysinit.target system-rsnapshot.slice mnt-backups.mount
```

Tip: Run `journalctl -u mnt-backups.mount` as root to verify that the automatic mounting and unmounting is taking place as expected.

How it works

We have a snapshot root under which all backups are stored. In this example, this is the directory `/mnt/backups/`. 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 /mnt/backups/
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 `/mnt/backups/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 does not 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.