

# Dirvish Guide by Jason Boxman (2005)



The original page is no longer available, so a copy it is hosted here.

## Introduction

### Intended Audience

System and network administrators and hobbyists who understand and appreciate the importance of data backup for disaster recovery, accidental filesystem operations, and intrusion forensics.

### Copyright and License

This document, Configuring and Using Dirvish for Snapshot Backups, is copyrighted © 2005 by Jason Boxman.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is available at <http://www.gnu.org/copyleft/fdl.html>.

Linux is a registered trademark of Linus Torvalds.

### Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies, that could be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility.

All copyrights are held by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

### Feedback

Please feel free to contact me with corrections, omissions, and questions: [jasonb@edseek.com](mailto:jasonb@edseek.com)

## New Versions of this Document

The newest version of this HOWTO will always first be made available at <http://wiki.edseek.com/>.

## Introspective: What's a Snapshot Backup?

Snapshot backups are a bit different than merely making copies of some files. With some filesystem trickery, each snapshot of a filesystem is nearly identical to the point in time the snapshot was taken. From here on, I will speak of filesystems, but you can substitute that for individual files, directories, or multiple filesystems of interest. Please note that the snapshot concept discussed here is different than the Linux Logical Volume Manager (LVM) concept of a snapshot, although the two can be used to compliment each other.

## Contrasting With Traditional Tape Paradigm

Traditional backup to tape, and frequently other storage devices today, revolves around performing a full backup of a filesystem. It's often not realistic to perform a full backup at every backup interval and especially so when that occurs nightly. Generally after a full backup, for some period of time thereafter incremental backups are performed, where only files that have changed are backed up. Performing a restore traditionally requires that the full backup nearest the point in time a restore is needed is utilized, followed by any incremental backups following it. Not fun.

Performing snapshot backups of a filesystem is almost a complete reversal. In essence, you take a full backup at every interval. However, only the files that have changed are actually backed up. What's more, only the differences between the files are transferred across the network infrastructure. Performing a restore merely requires finding the filesystem snapshot for the point in time you require a restore, and restoring.

## Anatomy of a Snapshot Backup

So just how does the magic work? The concept of a UNIX hard link is utilized. To explain, we must drop down to the filesystem level. Each unique file on the filesystem has a magic number, called an inode, which allows it to be referenced on disk. Each directory is simply a special file with a list of inodes that it contains mapped to filenames. As such, it's possible for any directory to have within its list more than one filename pointing to the same inode. Creating a new filename representation for an inode does not duplicate the actual data on disk the inode points to.

For example, pretend there is some file, called foo. Using the cp command, we can create a hard link to it, with a target filename of bar. Running stat on both filenames, we find that they both reference the same magic number. Accessing the inode by either associated filename allows you to modify the underlying data.

```
jasonb@faith:/tmp$ touch foo
jasonb@faith:/tmp$ stat foo
  File: 'foo'
```

```

  Size: 0          Blocks: 0          IO Block: 4096   regular empty
file
Device: dh/13d  Inode: 85784          Links: 1
...
jasonb@faith:/tmp$ cp -l foo bar
jasonb@faith:/tmp$ stat bar
  File: 'bar'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty
file
Device: dh/13d  Inode: 85784          Links: 2
...
jasonb@faith:/tmp$ ls -l foo bar
-rw-rw-r--  2 jasonb jasonb 0 Jan 30 16:43 foo
-rw-rw-r--  2 jasonb jasonb 0 Jan 30 16:43 bar

```

For a snapshot backup to work, the concept need merely be taken a step further. As indicated above, for hard links to work there has to be something to link against. The initial, full backup of a filesystem serves this purpose. The best way to demonstrate this is using rsync to backup a local filesystem, as it supports creating hard links in the manner we desire.

```

faith:/tmp# rsync -aH /var/ /tmp/var.0/
faith:/tmp# du -sh var.0/
158M    var.0/

```

Above, rsync was used to backup all data under /var to /tmp/var.0, ensuring that all attributes are preserved, including ownership, mode, and modification time.

After some additional system activity takes place, another filesystem backup is performed using rsync again. rsync has an option to create hard links to files under a different directory tree for files that have not changed. Our original backup in /tmp/var.0 is used for this purpose. Files that have changed are first unlinked, a critically important step, and then updated. Were changed files not unlinked first, the existing copy of the data would be updated and thus the original would be destroyed. Not good.

```

faith:/tmp# rsync -aH --delete --link-dest=/tmp/var.0 /var/ /tmp/var.1

```

The end result is two fold. First, each backup is the equivalent of a full filesystem backup, thanks to hard links. Second, the actual storage requirement for performing backups is greatly reduced, as only recently changed files occupy space.

```

faith:/tmp# du -h | grep 'var.[01]$'
158M    ./var.1
30M     ./var.0

```

Both directories contain an identical copy of /var, except for the files, directories, and attributes that have changed. The difference totals only 30MB.

## Practical Considerations

As hard linking is required, a traditional UNIX filesystem is a must. Snapshot backups work well on their own and as an intermediary between DVD-RAM or a tape library.

There are some important caveats worth mentioning, however.

- Either an individual hard disk or an array of disks is required.
- Relational Database Systems (RDBMS) cannot be backed up while live.
- The initial snapshot is bandwidth intensive, since all data must be copied the first time a snapshot is created.
- Likewise, if there is a high file creation or modification turnover rate, the storage space required for each new snapshot will be large. Such will be the case when backing up RDBMS dumps and log files.
- When the amount of storage space required exceeds that of a single hard disk, rotating a disk off-site is difficult.

## Installing Dirvish

Before we can start configuring Dirvish, we need to obtain and install it along with some dependencies. Optionally you can configure Dirvish to run nightly as a cronjob. You will probably do this after configuration, but it is presented in this section.

### Obtaining Needed Dependencies

Dirvish requires two Perl modules which may not ship with your Perl. You will need `Time::ParseDate` and `Time::Period`. You can install the needed modules via CPAN.

```
sarah:~# perl -MCPAN -e 'install Time::ParseDate'  
sarah:~# perl -MCPAN -e 'install Time::Period'
```

If you have never used CPAN before, you will be prompted with a list of questions regarding your system's setup. The defaults are acceptable in most cases. When selecting mirrors, list one or more of the numbers preceding the mirrors you want, separated by spaces.

If using CPAN is more Perl than you want to mess with, you can fetch the modules directly and install them by hand.

### Obtaining and Installing Dirvish

Installation is straightforward. If you are running Debian GNU/Linux, merely install the `dirvish` package with `apt-get install dirvish`. Otherwise, fetch the latest version of Dirvish from the official Web site. Unpack the tarball somewhere. Once that's done, read `INSTALL`. Finally, execute `install.sh`, which will ask you some questions.

```
root@faith:/usr/src/Dirvish-1.2# sh install.sh
perl to use (/usr/bin/perl)
What installation prefix should be used? ( ) /usr/local
Directory to install executables? (/usr/local/bin)
Directory to install MANPAGES? (/usr/local/man)
Configuration directory (/etc/dirvish) /usr/local/etc
Perl executable to use is /usr/bin/perl
Dirvish executables to be installed in /usr/local/bin
Dirvish manpages to be installed in /usr/local/man
Dirvish will expect its configuration files in /usr/local/dirvish
Is this correct? (no/yes/quit) yes
```

You will need to specify your preferred perl binary, a directory prefix, and a location for the configuration files. I choose `/usr/local`, as is customary, above. When you are happy with your choices, proceed.

```
Executables created.

Install executables and manpages? (no/yes) yes

installing /usr/local/bin/dirvish
installing /usr/local/bin/dirvish-runall
installing /usr/local/bin/dirvish-expire
installing /usr/local/bin/dirvish-locate
installing /usr/local/man/man8/dirvish.8
installing /usr/local/man/man8/dirvish-runall.8
installing /usr/local/man/man8/dirvish-expire.8
installing /usr/local/man/man8/dirvish-locate.8
installing /usr/local/man/man5/dirvish.conf.5

Installation complete
Clean installation directory? (no/yes) yes
Install directory cleaned.
```

Once installation is complete, you will have some shiny new Dirvish scripts available. The Debian package installs all the scripts, save `dirvish-locate`, under `/usr/sbin`, not `/usr/bin`.

## Setting up a cronjob

Once configuration is complete, covered later, you will likely want to run `dirvish-runall` every day. While you may wish to create a more complicated cronjob that mounts partitions or emails status information, you can use something as simple as the following. Running both `dirvish-expire` followed by `dirvish-runall` will ensure the expiration of images occurs before the actual backups begin.

```
0 0 * * * root /usr/local/bin/dirvish-expire --quiet ;
/usr/local/bin/dirvish-runall --quiet
```

You should always run `dirvish-expire` first. It is responsible for removing snapshots which have exceeded your retention policy time period, freeing up space you may need for the next run of

dirvish-runall. Ensure you allow enough time for dirvish-expire to finish.

## Configuring Dirvish for Snapshot Backups

To fully leverage Dirvish, we need to properly configure it for the scenarios within which it needs to operate. First, the master configuration must be defined. Next, a filesystem layout for storing backups must be determined. Then a complete configuration can be assembled. Finally, a few security features of Dirvish can be utilized.

### Reviewing the Master Configuration

Dirvish expects to find its master configuration file in `/etc/dirvish.conf` or `/etc/dirvish/master.conf`. The master configuration, referred to as `master.conf` from now on, contains global configuration selections which are used for all invocations of Dirvish scripts unless overridden later in more specific configuration files. Directives within the configuration either take a single value, a list of values, or a true or false value.

Single value directives appear on one line with one or more spaces after the colon. List value directives expect values on the succeeding lines following the colon, with at least one space of indentation preceding each item in the list. Boolean directives simply expect a 1 or 0 for true or false respectively following the colon and one or more spaces. Multiple instances of list directives will accumulate values, even from more specific configuration files.

```
# Single value
rsh: ssh -c arcfour

# Multivalue
exclude:
    /var/cache/apt/archives/*.deb
    .kde/share/cache/*
    .firefox/default/*/Cache/*

# Boolean
xdev: 0
```

Before you can usefully do anything, you must create a `master.conf`. Let's discuss some of the more interesting directives you will want to use, including `bank`, `image-default`, `log`, `index`, `xdev`, `exclude`, `expire-default`, and `Runall`.

First, you will want to define one or more banks. A bank is essentially a place where you store your backups. Each bank may have one or more vaults. A vault is the container for a single filesystem. You must create a vault for each filesystem you wish to backup.

```
bank:
    /snapshot/host-root
```

A vault is simply a directory within the root directory of the filesystem on a bank. You can choose any directory name you want, but something descriptive is usually best. Dirvish determines what is a

vault and what is not based on the presence of a dirvish/ subdirectory containing a file named `default.conf`. For example, you might have the following, with `/snapshot` defined as a bank and `host-root` defined as a vault.

```
/snapshot/host-root/dirvish/default.conf
```

There is no configuration option to define a vault. The presence of the `dirvish/default.conf` structure implicitly makes it a vault. Without it, it's just another unimportant directory.

A good candidate for a bank must have a large amount of available storage space. The longer your backup retention period, the more space you need. The more filesystems you backup, the more space you need. Dirvish will probe each bank in the order it is listed in `master.conf`.

Each snapshot, or image, within the vault must have a directory name. The default is specified with the `image-default`. Valid tokens are available in the *strftime(3)* manual. The following configuration

```
image-default: %Y%m%d-%H%M
```

will produce the directory below.

```
/snapshot/host-root/20080211-2203
```

All backups performed with Dirvish are logged. Each vault's snapshot has a log associated with it. You can choose to have this log, which includes all the output from running `rsync` with its verbose option, compressed with either *gzip* or *bzip2*. The former is a good choice. The default is to use no compression for the log file at all. You can specify either of the former two compression programs by using their respective names. For no compression, simply omit the directive `log` entirely or specify text.

```
log: bzip2
```

The above directive instructs Dirvish to compress each log file using the `bzip2` compression program.

A searchable index can be created by Dirvish after each snapshot has been taken, which is used later when you run `dirvish-locate` to search for a file you may wish to restore. If you do not specify this directive, no index will be created. You can use the same values as you can for `log`, above. Additionally, you can specify `none` if you wish for no index to be created, but do not want to omit including the directive for clarity.

```
index: none
```

The directive above entirely disables the creation of an index for `dirvish-locate`, meaning you will need to perform your own costly find from the command line.

Depending on your backup strategy, you may not want Dirvish to cross filesystem boundaries when performing backups. By default, the parameters passed to `rsync` do not instruct it to remain within a single mount point on the target filesystem. If you want to have a vault for each filesystem being backed up, you will want to enable the boolean option `xdev`. If not specified, the default action is to descend into all filesystems that may exist under the specified target.

```
xdev: 1
```

No mount points will be descended into with the above directive specified.

Finally, let us explore the exclude directive. Unsurprisingly it lets you prune files from the filesystem you may not want backed up. Pseudo filesystems, like /proc need not be backed up. You may also wish to exclude temporary files. File exclusion can be specified using all the regular expression rules used by rsync, covered in great detail in that program's manual.

```
exclude:  
  /etc/mtab  
  /var/lib/nfs/*tab  
  /var/cache/apt/archives/*.deb  
  .kde/share/cache/*  
  .firefox/default/*/Cache/*  
  /usr/src/**/*.  
  lost+found/
```

For example, the exclusion rules above exclude /etc/mtab and any files with a specific ending in /var/lib/nfs. It also excludes any Debian packages in /var/cache/apt/archives, all files in the K Desktop Environment's cache directory, and all files in Firefox's cache directory, with a wildcard to deal with the random filename it is given. Finally it uses a wildcard to match any directories under /usr/src that have object files and the specific directory lost+found/, where ever it may be. You can define additional excludes in vault specific configuration files for even more control.

Next, an expire-default is a wise idea. Otherwise, none of your snapshots will ever expire. Generally it is desirable for snapshots to be deleted eventually after being transferred to longer term storage media. expire-default is applied if no other expiration mechanism, such as expire-rule, matches a snapshot.

```
expire-default: +30 days
```

Last, you will need a Runall directive, which is a list of all vaults you want Dirvish to perform backups on when dirvish-runall is run. If you specify a time, the snapshot will have its modification time stamp changed to the specified time. If you specify a time that is in the future when dirvish-runall is run, your time will be forced 24 hours into the past.

```
Runall:  
  host-root 23:00  
  host-home 23:00
```

The specified time is entirely cosmetic. It gives the illusion of a snapshot being taken at some specific time for consistency. You can make it appear as if the backups actually run at 11 P.M. In either case, you can entirely omit a specified time.

## Bank and Vault Organizational Possibilities

There are a variety of ways you can choose to organize your backup filesystem layout. Your layout will depend on storage considerations, number of filesystems or machines you intend to backup, retention period, and personal preference. Briefly, let's look at two possible layouts.

If you have relatively few machines to backup, you may choose to create a bank for each machine. If



you are creating a snapshot for each individual filesystem for each machine, this setup makes it easier to group the snapshots together. If each machine has a filesystem for /usr, /home, and /var, the number of snapshots will quickly multiply. The layout might look like the output below:

```
/snapshot/host.example.com/host-root
/snapshot/host.example.com/host-home
/snapshot/other.example.com/other-root
/snapshot/other.example.com/other-home
```

If you have more than a handful of machines to backup, you may choose to create a single bank. If you perform a full snapshot across all relevant mount points (xdev: 0) a great many snapshots can live in a single bank. The layout may resemble the following:

```
/snapshot/host.example.com
/snapshot/other.example.com
/snapshot/mybox.example.com
```

The above is more likely to be a common deployment scenario and is what the author recommends.

## Assembling Your Configuration

Now we can create an actual master.conf. We will use a single bank and vault for each system.

```
bank:
  /snapshot/current

image-default: %Y%m%d
log: gzip
index: gzip

exclude:
  /var/cache/apt/archives/*.deb
  /usr/src/**/*.*
  lost+found/

Runall:
  host.example.com
  other.example.com

expire-default: +2 months
```

The above configuration defines a bank called /snapshot/current. An image-default in the style of four digit year, two digit month and and is followed by both the Dirvish log for the image and the image index being compressed with gzip. An exclude list includes some useful exclusions. Finally, Runall lists two vaults for dirvish-runall to operate on. The expire-default is a lengthy two months.

The configuration above will inspire a directory structure on the filesystem of each bank similar to that shown below.

```
/snapshot/current/host.example.com/dirvish/default.conf
/snapshot/current/other.example.com/dirvish/default.conf
```

By using `/snapshot/current`, it's possible to have another directory, `/snapshot/attic`, where vaults can be moved without `dirvish-expire` being able to see and remove them. If you remove a vault from the `Runall` directive, `dirvish-expire` will still probe your snapshot directory for any directories with a `dirvish/` subdirectory for image expiration. You have been warned.

Now we're almost ready to do some backups. First, however, we need to delve deeper into `default.conf` and tell `Dirvish` what we need it to do.

## Defining `default.conf`

The rest of `Dirvish`'s magic takes place in the configuration file for each individual vault, named `default.conf`. A vault exists, from the perspective of `Dirvish`, based on whether `default.conf` exists in a directory called `dirvish/` in a directory in a bank.

For example, for a vault named `other.example.com` to exist, the following directory structure would need to exist under a bank in `/snapshot`.

```
/snapshot/other.example.com/dirvish/default.conf
```

Inside our `default.conf`, we must define two key directives. First we will discuss `client` and then `tree`. Other directives discussed for `master.conf` can be redefined to override the global values when it makes sense to do so. (You wouldn't want to redefine `Runall` in `default.conf` for example.)

First, the `client` specifies the host target. `client` is the hostname, and possibly username, necessary to access the filesystem on the target machine specified by `tree`.

```
client: root@some.example.com
```

For example, the above `client` directive instructs `Dirvish` to use the system named `some` as the target and connect as the `root` user. If you omit the user, `Dirvish` will connect as the user running the `dirvish` binary, generally the `root` user.

```
client: 192.168.25.1
```

As demonstrated above, you can also specify the IP address of the target system.

Last, you must specify which path you want to backup with the `tree` directive. Generally this will be aligned with a filesystem boundary, but you can backup any path you wish.

```
client: root@sarah.example.com
tree: /
xdev: 0
index: gzip
```

The configuration above will backup the entire system, transcending mount points. Additionally, the `index` directive is specified which overrides the `Dirvish` default of not creating a list of files in each

snapshot. As mentioned earlier, you can override defaults set in master.conf as necessary in each default.conf.

Finally, you must initialize your vault. As this will make a complete copy of all the files under your tree, you may wish to perform this operation when it will not adversely effect the target system and network performance.

```
sarah:~# dirvish -vault other.example.com -init
```

```
sarah:~# ls /snapshot/other.example.com/  
20050323  dirvish
```

The command above will initialize the specified vault. If everything goes well, after some time the command will complete without incident. A shiny new snapshot should be available, your first.

## Backup Security Measures

You can restrict the permissions, which by default are more relaxed for images and log, index, and summary file. The meta-perm and image-perm options will allow you to specify your preferred permissions in octal. Best defined in master.conf, you can override the permissions if necessary in your vault configurations using default.conf.

```
image-perm: 700  
meta-perm: 600
```

Above, the filesystem permissions for each image are set to be readable only by the owner, generally the root user. Never fear, the permissions within the image itself will be the same as they were on the target system. Only the permissions on the image directory itself are changed. Additionally, the filesystem permissions for the files summary, log, and index are restricted. You will only have an index file if you enabled the index option.

Additionally, it's suggested that you limit access to each bank with restrictive filesystem permissions. You may wish to unmount the backup volume entirely when it's not being used, or remount it read only.

## Configuring Your Transport: ssh and local

### ssh Transport

The default transport for Dirvish is OpenSSH, which brings you encryption and remote command execution. However, generally by default OpenSSH is configured to use passwords. If you intend to run dirvish-runall via cron, you will want to configure support for public key authentication instead on each target machine.

Configuring OpenSSH to use public key authentication is fairly straightforward. On the system that will be running dirvish-runall, the backup server, you need to generate a public key for the root user. For that, run ssh-keygen. You could use a different, less privileged user, but then you cannot read all files

on the target.

```
root@backup:~# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
6b:1f:aa:93:17:cc:23:22:05:ff:f1:22:2e:f7:0b:6c root@backup
```

If you enter an empty, or null, passphrase you will not be prompted to enter your passphrase when you connect using public key authentication. This is a popular method of handling connections to remote machines using OpenSSH. However, you can still choose to use a passphrase and employ a tool like keychain to cache your passphrase, even after you logout.

Once that's complete, you will have two different keys. The first is your private key, `id_rsa`, which you should guard with your life. This key should never, ever be distributed. The second key, `id_rsa.pub`, is the public key. Each target host you wish to connect to using public key authentication must have a copy of this key distributed to it. You can even do this with `ssh` from the target machine.

More recently, you'd use `ssh-keygen -t dsa` to generate a DSA key instead. Everything about RSA key distribution and creation applies to DSA keys, too.

```
root@target:~# ssh root@backup 'cat ~/.ssh/id_rsa.pub' >>
~/.ssh/authorized_keys
root@target:~# chmod 600 ~/.ssh/authorized_keys
root@target:~# chmod 700 ~/.ssh/
root@target:~# chmod go-w ~
```

Executing the above on the target machine should append a copy of the RSA public key from the backup server to the `authorized_keys` file, a prerequisite for public key authentication to function. Additionally, your filesystem permissions must be correct or OpenSSH will refuse public key authentication requests. `~/.ssh/` must be accessible only to the root user along with the `authorized_keys` file. Additionally the home directory for root, referred to as `~` above, must not be world or group writable.

With that complete, you should be able to successfully login to the target from the backup server running Dirvish. If you chose not to use a passphrase when you generated your RSA key pair, you will be immediately logged into the target. If you entered a passphrase, you must enter it as you would for a password authentication login.

```
root@backup:~# ssh root@target
...
Last login: Mon Feb 28 18:24:51 2005 from backup.example.com
root@backup:~#
```

If public key authentication doesn't work and instead you are immediately prompted for your password, your directory permissions may be incorrect on the target. You can disable `StrictModes` in `/etc/ssh/sshd_config` or the equivalent file on your system and restart OpenSSH to quickly verify if it's

a permissions issue. In addition to what `sshd_config(5)` states, if the user directory is group writable, public key authentication will fail.

Repeat this procedure for each host you wish to access over the network using the ssh transport, the default used with Dirvish.

The backup server essentially has the keys to the city. Ensuring it is kept secure is crucial. Of course, it also has all your backups, so that goes without saying.

## local Transport

Another option, the local transport is merely the absence of using any network transport. The local transport is invoked when your client's hostname is the same as that reported by the hostname command.

```
sarah:~# hostname  
sarah.example.com
```

Your `default.conf` would then include that hostname for the client option.

## Using Advanced Dirvish Features

Unsurprisingly, Dirvish supports a variety of more advanced features that can be utilized once your initial configuration is tuned to your liking. Dirvish supports executing helper scripts at each point during the actual snapshot process for each vault. You can configure complex expiration rules in your `master.conf` and perform complex queries using `dirvish-locate` to track down individual files in your snapshots needed for recovery.

## Using pre- and post- Scripts

Often there are some tasks you may wish to perform prior to or immediately after creating a snapshot for a particular vault. Fortunately, Dirvish cleanly integrates the execution of helper scripts.

You have four hooks available to you so your script can be run at the most effective point during the backup process. Each script is executed using `/bin/sh`. The pre-server hook is executed first on the system running Dirvish. Next, pre-client is executed on the target system. If the command is the path to a script, it must exist on the target system. Next, post-client is executed on the target system after `rsync` has completed. Finally, post-server is executed on the system running Dirvish.

Each script has access to several environment variables.

```
DIRVISH_SERVER=backup.example.com  
DIRVISH_CLIENT=target.example.com  
DIRVISH_SRC=/  
DIRVISH_DEST=/snapshot/target.example.com/20050316/tree  
DIRVISH_IMAGE=target.example.com:default:20050316
```

```
DIRVISH_EXCLUDE=/snapshot/target.example.com/20050316/exclude
```

The first two are the name of the server and client respectively, the first generated from the hostname command. \$DIRVISH\_SRC is the source directory and serves as the current working directory on the target system. \$DIRVISH\_DEST is the destination directory on the backup server, and the working directory for scripts executing on the backup server. \$DIRVISH\_IMAGE includes the vault name, the branch name, usually just default, and the directory name for the snapshot. Finally, \$DIRVISH\_EXCLUDE contains the path to the exclude file when executing pre-server, in case you wish to modify it. If no exclude list is defined anywhere, \$DIRVISH\_EXCLUDE will not be defined.

Due to the manner in which the argument to the above four hooks is passed, you can only provide commands of limited complexity. You can use double quotes, but not single quotes. It's still possible to accomplish much without creating additional scripts, though.

```
pre-client: /usr/sbin/invoke-rc.d fetchmail stop
```

For example, the above shell command simply shuts down a service. The shell command is executed on the remote system using ssh unless you defined rsh to be some other rsh compatible remote shell. pre-client and the other three hook commands are single value directives, so your shell command must be on a single line.

```
post-server: ; /usr/bin/ssh target "/sbin/sfdisk -d /dev/sda" >
$DIRVISH_DEST/../sfdisk.out
```

Another example, the above shell command is executed after the backup has been completed and all other hooks have executed. It initiates a connection using OpenSSH and runs sfdisk on the target system to retrieve the partition table for /dev/sda. The environment variable \$DIRVISH\_DEST is used to save the output from the command on the backup server, one level up from the directory where the backup has been saved. The leading semicolon is necessary for the binary that follows, in this case ssh, to see the environment variables set by Dirvish. (If it isn't working, the semicolon can be removed.)

If you need to backup a relational database, you have several options. You can simply stop it, as was done above with fetchmail, or dump the entire database and transfer it either as part of a post-server shell command or using rsync itself by dumping it somewhere on the target system during the pre-server or pre-client phases.

```
post-server: ; /usr/bin/ssh target "mysqldump -A -a -e" >
$DIRVISH_DEST/../mysql.%Y%m%d
```


A mysqldump of all databases is created and saved in the parent directory of \$DIRVISH\_DEST after the snapshot has been taken when the above shell command is executed by the post-server hook. You will notice date tokens are parsed before the command is executed. Any option listed in the strftime(3) manual can be used, as with image-default directive discussed earlier.

```
pre-client: ; /usr/bin/mysqldump -A -a -e > $DIRVISH_SRC/mysql.%Y%m%d
post-client: ; /bin/rm -f $DIRVISH_SRC/mysql.%Y%m%d
```

Another possibility, mysqldump is run from the pre-client hook and the dump is removed from the post-client hook thereafter. The above might be useful if you keep a directory of database dumps on the target and wish to perform the dump immediately prior to taking a snapshot of the filesystem where

the dumps reside. In that instance, you would want to omit the post-client that deletes the dump from \$DIRVISH\_SRC.

Ralf Weinedel suggests the following modification if your distribution is using the dash shell for /bin/sh. The change is made around line 942.



```
if (${dir} !~ /^:/)
{
    #srcmd = sprintf ("%s 'cd %s; %s %s' >>%s",
    srcmd = sprintf ("%s 'cd %s; export %s %s' >>%s",
        ("${shell}" || "/bin/sh -c"),
        ${dir}, ${env},
        $cmd,
        ${log}
    );
} else {
    #srcmd = sprintf ("%s '%s %s' >>%s",
    srcmd = sprintf ("%s 'export %s %s' >>%s",
        ("${shell}" || "/bin/sh -c"),
        ${env},
        $cmd,
        ${log}
    );
}
```

## Using Expire Rules to Tweak Snapshot Lifetime

Often, you may wish to more precisely control snapshot expiration. Dirvish allows you to express complex expiration rules in either crontab(1) or Time::Period format. The underlying parser will be Time::Period, irrespective of which format you prefer when specifying your rules. The directive, which expects a list, is expire-rule.

The biggest caveat is the manner in which time ranges are specified. For example, the nine hour period between 9 a.m. and 6 p.m. would be specified thusly in Time::Period format:

```
hour { 9am-5pm }
```

You will note the end of the range includes all moments through 5:59 p.m. If 6 p.m. was specified instead, the time through 6:59 p.m. would be included and the range would be off by one hour. When specifying minutes, the same rule applies.

To keep snapshots created every Thursday for 15 days, snapshots created on the 28th of every month for six months, snapshots created every afternoon from 1 p.m. until 3 p.m. forever, and all other snapshots for 30 days, you might have a list of rules like the following.

```
expire-rule:
wday { thur } +15 days
```

```
mday { 28 } +6 months
hour { 1pm-2pm } never
expire-default: +30 days
```

Your rules can be as complex as necessary. Ordering is significant. When multiple rules match a snapshot, the last matching rule wins. You can specify expiration rules in default.conf in addition to master.conf to suit your needs. Once you have accounted for significant snapshots with expiration rules, use an expire-default to catch the rest. Snapshots are not deleted until you run dirvish-expire.

## Locating Files for Restore with dirvish-locate

Dirvish offers a quick way of searching for files when you need to perform a restore of one or more individual files. The dirvish-locate script will search through indices created by the index directive, eliminating the need for a costly find across many snapshots.

You can use Perl's rich regular expression syntax to describe the file you are searching for. The results will include the modification time for each match and which snapshots, listed by snapshot name, contain the match. This is useful for tracking down a file when you have some idea how old or new it must be to restore a good copy.

```
sarah:~# dirvish-locate faith-var 'http\.us'
1 matches in 30 images
/var/lib/apt/lists/http.us.debian.org_debian_dists_unstable_non-
free_source_Sources
  Mar 18 15:19 20050320, 20050319, 20050318
  Mar 14 15:23 20050317, 20050316, 20050315, 20050314
  Mar 12 15:16 20050313
  Mar  1 15:15 20050312, 20050311, 20050310, 20050309, 20050308, 20050307
                20050306, 20050305, 20050304, 20050303, 20050302, 20050301
  Feb 27 15:15 20050228
  Feb 26 15:15 20050227, 20050226
  Feb 25 15:25 20050225
  Feb 23 15:21 20050224, 20050223
  Feb 21 15:23 20050222, 20050221
  Feb 19 15:15 20050220, 20050219
```

To use dirvish-locate, you must specify the specific vault you wish to perform your search against. Following that, you must specify an expression to match the file you are looking for. To start, it can be as simple as the filename itself. Since Perl's regular expression syntax is used, the expression above has the period, a special character, escaped with a backslash. You should enclose your expression in single quotes, unless it is merely a filename, so your shell does not process any meta characters.

You will definitely need to specify some basic expression if dirvish-locate warns you that your string returns too many results.

```
sarah:~# dirvish-locate faith-usr 'gnucash'
199 matches in 30 images
Pattern 'gnucash' too vague, listing paths only.
/usr/bin/gnucash
/usr/bin/gnucash-config
```



...

In the example above, a simple search for any files containing the string gnucash was attempted. Unfortunately, the string was too vague. There were too many matches to reasonably return specific information on each result. Fortunately, the results contain a list of files that did match so you can narrow your results. For example, perhaps we are interested in only the file /usr/bin/gnucash.

```
sarah:~# dirvish-locate faith-usr '^/usr/bin/gnucash'
2 matches in 30 images
/usr/bin/gnucash
  Aug 18  2004 20050320, 20050319, 20050318, 20050317, 20050316, 20050315
                20050314
/usr/bin/gnucash-config
  Aug 18  2004 20050320, 20050319, 20050318, 20050317, 20050316, 20050315
                20050314
```

Using the regular expression above, the results are narrowed down to something much more manageable. If you have not used POSIX or Perl regular expressions, the carrot symbol might be unfamiliar in the context above. The symbol, when used at the beginning of an expression, tells dirvish-locate that the string that follows must exist at the beginning of any match. You will notice above, both /usr/bin/gnucash and /usr/bin/gnucash-config matched.

```
sarah:~# dirvish-locate faith-usr '.*gnucash'
3 matches in 30 images
/usr/bin/gnucash
  Mar 15 18:37 20050321
  Aug 18  2004 20050320, 20050319, 20050318, 20050317, 20050316, 20050315
                20050314
/usr/lib/gnucash/overrides/gnucash
  Mar 15 18:37 20050321
  Aug 18  2004 20050320, 20050319, 20050318, 20050317, 20050316, 20050315
                20050314
/usr/lib/menu/gnucash
  Mar 15 18:32 20050321
  Aug 18  2004 20050320, 20050319, 20050318, 20050317, 20050316, 20050315
                20050314
```

Above, the expression uses the same string as before, but with a period and an asterisk at the beginning. When you include .\*, you allow any number of characters to occupy that space in the result string. Above, you can see a variety of characters precede the string gnucash.

```
sarah:~# dirvish-locate faith-usr '/usr/bin/gnucash$'
1 matches in 30 images
/usr/bin/gnucash
  Aug 18  2004 20050320, 20050319, 20050318, 20050317, 20050316, 20050315
                20050314
```

As demonstrated above, when a dollar sign is at the end of a regular expression, it tells dirvish-locate that any match must end with the string that precedes the dollar sign.

For detailed instructions on using Perl's regular expression syntax, refer to the documentation

perlretut on a terminal near you.

```
jasonb@faith:~$ perldoc perlretut
```

## When Good Backups Go Bad

Once in a while, a backup will not go as planned.

### fatal error (12) -- filesystem full

If there isn't sufficient disk space for a backup to complete, even after dirvish-expire has run, the status file in the vault's snapshot directory will include useful diagnostic output. Combined with the `rsync_error` file, if one is present, it is possible to deduce what consumed all the available disk space.

```
client: root@nebula.example.com
tree: /home/shared
rsh: ssh -c blowfish
Server: sarah.example.com
Bank: /snapshot/current
vault: nebula-shared
branch: default
Image: 20080405
Reference: 20080404
Image-now: 2008-04-05 23:00:00
Expire: +60 days == 2008-06-04 23:00:00
exclude:
    /proc/*
    /sys/*
    /tmp/*
    /etc/mtab
    /var/lib/nfs/*tab
    /var/cache/apt/archives/*.deb
    /var/cache/apt/archives/partial/*.deb
    /usr/src/**/*.*ko
    lost+found/
    *~
    .nfs*
SET permissions devices numeric-ids stats
UNSET checksum init sparse whole-file xdev xzfer

ACTION: rsync -vrltH --delete -pgo --stats -D --numeric-ids --exclude-
from=/snapshot/current/nebula-shared/20080405/exclude --link-
dest=/snapshot/current/nebula-shared/20080404/tree
root@nebula.example.com:/home/shared/ /snapshot/current/nebula-
shared/20080405/tree
Backup-begin: 2008-04-06 05:31:11
Status: fatal error (12) -- filesystem full
```

Above, the status file contents for a failed backup image. The Reference is the image that is passed to the `-link-dest` option to `rsync`, as discussed earlier. Dirvish will always attempt to use the last known successful backup as the Reference for `-link-dest`. Otherwise, a failed backup may result in future failed backups by breaking your chain of hardlinked backups.

By examining the `rsync_error` file, the actual failure message can be determined.

```
*** Execution cycle 0 ***

rsync: write failed on "/snapshot/current/nebula-
shared/20080405/tree/big.iso": No space left on device (28)
rsync error: error in file IO (code 11) at receiver.c(252) [receiver=2.6.8]
rsync: connection unexpectedly closed (5652802 bytes received so far)
[generator]
rsync error: error in rsync protocol data stream (code 12) at io.c(463)
[generator=2.6.8]
```

Examining `20080405/log` will reveal the file being transferred at the time of failure. At this point, it may be necessary to

- Add a file or directory pattern to the exclude list in either `master.conf` or the vault's `default.conf`
- Increase the available disk space on the snapshot volume
- Adjust the image retention policy and manually `rm -Rf` some image directories

## warning (24) -- file vanished on sender

Sometimes, transient files can cause a backup failure. It's treated as a warning, but the summary file Status field will not be success even if the backup otherwise completes successfully. Investigate `log.gz` and add the necessary exclude for transient files.

```
file has vanished: "/var/log/ntpstats/loopstats.20081220"
```

Thereafter, you can simply edit the summary file and change Status to indicate success.

```
root@backup:/snapshot/vault# ls -l */summary | \
xargs -l1 -i% perl -i -pe 's/^Status.*warning.*$/Status: success/g' %
```

## Recovering From a Failed Backup Attempt

Depending on the length of time required to complete a backup, it may be quite unfortunate to discover a failure. It is, however, possible to recover from a filesystem full failure without completely rerunning a backup. Instead, the following solution often works.

First, find the last two known good backups. Next, for the current failed backup, locate the summary file. Inside is the exact command used to start the `rsync` process. Copy that command and be prepared to execute it as the backup user, probably root, with a few changes.

```
root@backup:/snapshot# rsync -vrltH --delete -pgo --stats -D --numeric-ids
```

```
\
--exclude-from=/snapshot/current/nebula-shared/20080405/exclude \
--link-dest=/snapshot/current/nebula-shared/20080404/tree \
root@nebula.example.com:/home/shared/ /snapshot/current/nebula-
shared/20080405/tree
```

Because of the way the `-link-dest` option works, it is necessary for there to be two known good snapshot instances to work with. Generally, use the last known good image as the path for `-link-dest`. Then, use the next known good image path as the source for the `rsync` command. Finally, the target will be the path to the failed snapshot directory. Additionally, the `-exclude-from` must be removed, since the file does not exist and the last good backups already excluded the files.

```
root@backup:/snapshot# rsync -vrltH --delete -pgo --stats -D --numeric-ids
\
--link-dest=/snapshot/current/nebula-shared/20080404/tree \
/snapshot/current/nebula-shared/20080403/tree/ /snapshot/current/nebula-
shared/20080405/tree
```



If disk space has been exhausted because the hardlink chain to some files in prior snapshots is broken, simply substitute the last good backup with the last good snapshot that includes the missing files. In such a fashion, the consumed disk space can be recovered without deleting the entire snapshot history.

Successful backup images are relatively simple to locate.

```
sarah:/snapshot/current/nebula-shared# grep -i success 2008040[34]/summary
20080403/summary:Status: success
20080404/summary:Status: success
```

If you do not have two successful images, or you wish to use the same image for both the `-link-dest` option and the source of the `rsync` command, you must use `cp -al` to make an extra, hardlink only copy of your single snapshot.

```
sarah:/snapshot/current/nebula-shared# cp -al 20080404/tree
/snapshot/temp_image
```



Remember to remove it later, as it will begin to consume space as it starts to contain files with inodes that no longer exist anywhere else due to `dirvish-expire` expiring images!

Finally, you must edit the summary file and change `Status` to indicate success, the special keyword `Dirvish` uses to find the last good snapshot. Simply edit the summary file and change the failure message to the word success.

```
Status: success
```

Afterward, future backups should function as expected.

## Links and Resources

Mike Rubel's [Easy Automated Snapshot-Style Backups with Linux and Rsync](#)

Explanation of [Hard Links](#) at the Portland Pattern Repository

In Common threads: [OpenSSH key management](#), Part 2 Gentoo's Daniel Robbins explains using Keychain

[OpenSSH Public Key Authentication](#)

From:

<https://wiki.diala.net/> - **Diala Wiki**

Permanent link:

<https://wiki.diala.net/doc:boxman>

Last update: **27.08.2023 18:02**

