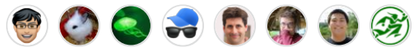

NAME

znapzendzetup - znapzend setup utility

SYNOPSIS

znapzendzetup *command* [*common-options...*] [*command-options...*]

where 'common-options' (must precede command-options) is a mix of the following:

```
[--rootExec={pfexec|sudo}] \
[--features={x,y,...}] \
    {lowmemRecurse zfsGetType} \
[--debug]
```

and where 'command' and its unique options is one of the following:

```
create  [--recursive] [--mbuffer=<path>[:<port>]] [--mbuffersize=<size>] \
        [--pre-snap-command=<command>] \
        [--post-snap-command=<command>] \
        [--tsformat=<format>] --donotask \
        [--send-delay=<time>] \
        SRC plan dataset \
        [ DST[:key] plan [[user@]host:]dataset
            [pre-send-command] [post-send-command] ]
```

```
delete  [--dst=key] <src_dataset>

edit    [--recursive=on|off] \
        [--mbuffer=<path>[:<port>]|off] [--mbuffersize=<size>] \
        [--pre-snap-command=<command>|off] \
        [--post-snap-command=<command>|off] \
        [--tsformat=<format>] --donotask \
        [--send-delay=<time>] \
        SRC [plan] dataset \
        [ DST:key [plan] [dataset] \
            [pre-send-command|off] [post-send-command|off] ]

edit    <src_dataset>

enable  <src_dataset>

disable <src_dataset>

enable-dst  <src_dataset> <DST_key>

disable-dst <src_dataset> <DST_key>

list    [--recursive] [--inherited] [src_dataset...]

export  <src_dataset>

import  [--write] [--prop <property>=<value>, [--prop ...] ...]
        <src_dataset> [<prop_dump_file>]

help

man
```

DESCRIPTION

Use znapzendsetup to configure your backup tasks. The cli is modled after the zfs commandline.

After modifying the configuration, send a HUP signal to your znapzend daemon for it to re-read the configuration.

Common options include:

--features=feature1,feature2,...
: enables a few enhanced zfs module features to be on par with those used by the znapzend command (but far fewer are relevant for the znapzendzetup)

  Available features:

  lowmemRecurse

use 'lowmemRecurse' on systems where you have too many datasets, so a recursive listing of attributes to find backup plans exhausts the memory available to `znapzend(zetup)`: instead, go the slower way to first list all impacted dataset names, and then query their configs one by one.

zfsGetType
use 'zfsGetType' if your 'zfs get' supports a '-t' argument for filtering by dataset type at all (e.g. one in Solaris 10 does not), AND lists properties for snapshots by default when recursing (e.g. the one in Solaris 10u8 already does), so that there is too much data to process while searching for backup plans.

If these two conditions apply to your system, the time needed for a '--recursive' listing of backup plans can literally differ by hundreds of times (depending on the amount of snapshots in that dataset tree... and a decent backup plan will ensure you have a lot of those), so you would benefit from requesting this feature.

This feature should not impact the default (non- '--recursive') listings however.

Below a few notes on main commands.

create

The heart of the znapzend backup is the plan. The plan specifies how often to backup and for how long to keep the backups. A plan is required both for the source and the destination datasets.

The plan consists of a series of retention periods to interval associations:

```
retA=>intA,retB=>intB,...
```

Both intervals and retention periods are expressed in standard units of time or multiples of them. You can use both the full name or a shortcut according to the following table:

```
second|sec|s
minute|min
hour|h
day|d
week|w
month|mon|m
year|y
```

To keep one copy every 30 minutes for one week, specify:

```
1week=>30min
```

To keep one copy every two days for 10 years:

```
10year=>2day
```

In a minimal setup, you just specify a plan for the **SRC** fileset. This will cause snapshots to be taken and destroyed according to the plan. You can then add one or several destinations (**DST**) both local (preferably on a different pool) or remote.

When adding multiple **DST** entries, each will get labeled for later identification, optionally you can specify your own label.

--tsformat=limited-strftime-format
The **--tsformat** option specifies how the names of the snapshots are constructed.

The syntax is strftime-like. The string must consist of the mandatory

```
%Y %m %d %H %M %S %z
```

Optionally,

`-_.:`

characters as well as any alphanumeric character are allowed.

If not specified, **--tsformat** defaults to `%Y-%m-%d-%H%M%S` .

If **--tsformat** string is suffixed by a 'Z', times will be in UTC. E.g.:

```
--tsformat='%Y-%m-%dT%H:%M:%SZ'
```

NOTE: that windoz will probably not like the `:` characters. So if you intend to browse the snapshots with windoz, you may want to use a different separator.

--mbuffer=/usr/bin/mbuffer
Specify the path to your copy of the mbuffer utility.

--mbuffer=/usr/bin/mbuffer:31337
Specify the path to your copy of the mbuffer utility and the port used on the destination. Caution: znapzend will send the data directly from source mbuffer to destination mbuffer, thus data stream is **not** encrypted.

--mbuffersize=number{b|k|M|G}
The size of the mbuffer can be set with the **--mbuffersize** option. It supports the following units:

```
b, k, M, G
```

To specify a mbuffer size of 100MB:

```
--mbuffersize=100M
```

If not set, the buffer size defaults to 1GB.

--donotask
Apply changes immediately. Without being asked if the config is as you intended it to be.

--pre-snap-command=/path/bin args, --post-snap-command=/path/bin args
Run commands/scripts before and after snapshots are taken on source. e.g. for database locking/flushing (pre) and unlocking (post).

If you deal with a mariadb/mysql database, you can use

```
pre-snap-command  = /opt/oep/mariadb/bin/mysql -e "set autocommit=0;flush tables with read lock;\\! /bin/sleep 600"
&  /usr/bin/echo $! > /tmp/mariadblock.pid ; sleep 10
post-snap-command = /usr/bin/kill `/usr/bin/cat /tmp/mariadblock.pid`;/usr/bin/rm /tmp/mariadblock.pid
```

to make sure that the on-disk data is consistent when snapshotting. The lock stays in place only for the duration of the lingering connection to mysql we need to employ, or until the snapshotting attempt times out. For this to work, add the root password of your mariadb/mysql database setup into ~root/.my.cnf and make sure the file permissions are tight ...

The pre and post snapshot commands can find the name and time of the snapshot in the environment variables *ZNAP_NAME* and *ZNAP_TIME*.

--send-delay
Specify delay (in seconds) before sending snaps to the destination. May be useful if you want to control sending time.

pre-send-command=/path/bin args, post-send-command=/path/bin args
Run command/script before and after sending the snapshot to the destination. Intended to run a remote script via ssh on the destination, e.g. to bring up a backup disk or server. Or to put a zpool online/offline:

```
"ssh root@bserv zpool import -Nf tank"
"ssh root@bserv zpool export tank"
```

delete

to remove configuration from a dataset just give its name

```
znapzendzetup delete I<dataset>
```

the **delete** function understands the following options:

--dst=key
to only remove a destination, specify the key of the destination. Use the **list** function to see the keys.

edit

modify the configuration of a dataset. See the descriptions in the **create** function for details.

If **edit** is used with a source dataset as single argument, properties can be edited in an editor.

export

dumps the backup configuration of a dataset

```
znapzendzetup export I<dataset>
```

import

reads configuration data from a file or STDIN and prints it content

--write
actually store the new configuration into the dataset given on the command line.

--prop key="value" [--prop ...]
may be called multiple times to override properties in the imported config.

EXAMPLES

create a complex backup task

```
znapzendzetup create --recursive --mbuffer=/opt/omni/bin/mbuffer \
   --mbuffersize=1G --tsformat='%Y-%m-%d-%H%M%S' \
   --pre-snap-command="/bin/sh /usr/local/bin/lock_flush_db.sh" \
   --post-snap-command="/bin/sh /usr/local/bin/unlock_db.sh" \
   SRC '7d=>1h,30d=>4h,90d=>1d' tank/home \
   DST:a '7d=>1h,30d=>4h,90d=>1d,1y=>1w,10y=>1month' backup/home \
   DST:b '7d=>1h,30d=>4h,90d=>1d,1y=>1w,10y=>1month' \
      root@bserv:backup/home \
      "/root/znapzend.sh dst_b pool on" \
      "/root/znapzend.sh dst_b pool off"
```

copy the setup from one fileset to another

```
znapzendzetup export tank/home | znapzendzetup import --write tank/new_home
```

RUNNING AS AN UNPRIVILEGED USER

In order to allow a non-privileged user to use it, the following permissions are required on the ZFS filesystems:

Sending end: *destroy,hold,mount,send,snapshot,userprop* Receiving end: *create,mount,receive,userprop*

COPYRIGHT

LICENSE

AUTHOR

Tobias Oetiker <tobi@oetiker.ch> Dominik Hassler <hadfl@cpan.org>

HISTORY

2016-09-23 ron Destination pre and post send/receive commands 2014-07-22 had Pre and post snapshot commands 2014-06-29 had Flexible snapshot time format 2014-06-01 had Multi destination backup 2014-05-30 had Initial Version