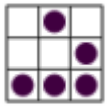

Aaron Toponce

{ 2013 01 02 }

ZFS Administration, Part XVI- Getting and Setting Properties

Table of Contents

Motivation

Just as with Zpool properties, datasets also contain properties that can be changed. Because datasets are where you actually store your data, there are quite a bit more than with storage pools. Further, properties can be inherited from parent datasets. Again, not every property is tunable. Many are read-only. But, this again gives us the ability to tune our filesystem based on our storage needs. One aspect with ZFS datasets also, is the ability to set your own custom properties. These are known as "user properties" and differ from "native properties".

Because there are just so many properties, I've decided to put the administration "above the fold", and put the properties, along with some final thoughts at the end of the post.

Getting and Setting Properties

As with getting and setting storage pool properties, there are a few ways that you can get at dataset properties as well- you can get all properties at once, only one property, or more than one, comma-separated. For example, suppose I wanted to get just the compression ration of the dataset. I could issue the following command:

```
# zfs get compressratio tank/test
NAME       PROPERTY       VALUE  SOURCE
tank/test  compressratio  1.00x  -
```

If I wanted to get multiple settings, say the amount of disk used by the dataset, as well as how much is available, and the compression ratio, I could issue this command instead:

```
# zfs get used,available,compressratio tank/test
tank/test  used                  1.00G                  -
tank/test  available             975M                   -
tank/test  compressratio         1.00x                  -
```

And of course, if I wanted to get all the settings available, I could run:

```
# zfs get all tank/test
NAME       PROPERTY              VALUE                  SOURCE
tank/test  type                  filesystem             -
tank/test  creation              Tue Jan  1  6:07 2013  -
tank/test  used                  1.00G                  -
tank/test  available             975M                   -
tank/test  referenced            1.00G                  -
```

```
tank/test  compressratio         1.00x                  -
tank/test  mounted               yes                    -
tank/test  quota                 none                   default
tank/test  reservation           none                   default
tank/test  recordsize            128K                   default
tank/test  mountpoint            /tank/test             default
tank/test  sharenfs              off                    default
tank/test  checksum              on                     default
tank/test  compression           lzjb                   inherited from tank
tank/test  atime                 on                     default
tank/test  devices               on                     default
tank/test  exec                  on                     default
tank/test  setuid                on                     default
tank/test  readonly              off                    default
tank/test  zoned                 off                    default
tank/test  snapdir               hidden                 default
tank/test  aclinherit            restricted             default
tank/test  canmount              on                     default
tank/test  xattr                 on                     default
tank/test  copies                1                      default
tank/test  version               5                      -
tank/test  utf8only              off                    -
tank/test  normalization         none                   -
tank/test  casesensitivity       sensitive              -
tank/test  vscan                 off                    default
tank/test  nbmand                off                    default
tank/test  sharesmb              off                    default
```

```
tank/test  refquota             none                  default
tank/test  refreservation       none                  default
tank/test  primarycache         all                   default
tank/test  secondarycache       all                   default
tank/test  usedbysnapshots      0                     -
tank/test  usedbydataset        1.00G                 -
tank/test  usedbychildren       0                     -
tank/test  usedbyrefreservation 0                     -
tank/test  logbias              latency               default
tank/test  dedup                off                   default
tank/test  mlslabel             none                  default
tank/test  sync                 standard              default
tank/test  refcompressratio     1.00x                 -
tank/test  written              0                     -
```

Inheritance

As you may have noticed in the output above, properties can be inherited from their parents. In that case, I set the compression algorithm to "lzjb" on the storage pool filesystem "tank" ("tank" is more than just a storage pool- it is a valid ZFS dataset). As such, any datasets created under the "tank" dataset will inherit that property. Let's create a nested dataset, and see how this comes into play:

```
# zfs create -o compression=gzip tank/test/one
# zfs get -r compression tank
```

```
NAME           PROPERTY     VALUE     SOURCE
tank           compression  lzjb      local
tank/test      compression  lzjb      inherited from tank
tank/test/one  compression  gzip      local
```

Notice that the "tank" and "tank/test" datasets are using the "lzjb" compression algorithm, where "tank/test" inherited it from its parent "tank". Whereas with the "tank/test/one" dataset, we chose a different compression algorithm. Let's now inherit the parent compression algorithm from "tank", and see what happens to "tank/test/one":

```
# zfs inherit compression tank/test/one
# zfs get -r compression tank
NAME           PROPERTY     VALUE     SOURCE
tank           compression  lzjb      local
tank/test      compression  lzjb      inherited from tank
tank/test/one  compression  lzjb      inherited from tank
```

In this case, we made the change from the "gzip" algorithm to the "lzjb" algorithm, by inheriting from its parent. Now, the "zfs inherit" command also supports recursion. I can set the "tank" dataset to be "gzip", and apply the property recursively to all children datasets:

```
# zfs set compression=gzip tank
# zfs inherit -r compression tank/test
# zfs get -r compression tank
NAME           PROPERTY     VALUE     SOURCE
tank           compression  gzip      local
```

```
tank/test      compression  gzip      inherited from tank
tank/test/one  compression  gzip      inherited from tank
```

Be very careful when using the "-r" switch. Suppose you quickly typed the command, and gave the "tank" dataset as your argument, rather than "tank/test":

```
# zfs inherit -r compression tank
# zfs get -r compression tank
NAME           PROPERTY     VALUE     SOURCE
tank           compression  off       default
tank/test      compression  off       default
tank/test/one  compression  off       default
```

What happened? All compression algorithms got reset to their defaults of "off". As a result, be very fearful of the "-r" recursive switch with the "zfs inherit" command. As you can see here, this is a way that you can clear dataset properties back to their defaults, and apply it to all children. This applies to datasets, volumes and snapshots.

User Dataset Properties

Now that you understand about inheritance, you can understand setting custom user properties on your datasets. The goal of user properties is for applications designed around ZFS specifically, to take advantage of those settings. For example, poudriere is a tool for FreeBSD designed to test package production, and to build FreeBSD packages in bulk. If

using ZFS with FreeBSD, you can create a dataset for poudriere, and then create some custom properties for it to take advantage of.

Custom user dataset properties have no effect on ZFS performance. Think of them merely as "annotation" for administrators and developers. User properties must use a colon ":" in the property name to distinguish them from native dataset properties. They may contain lowercase letters, numbers, the colon ":", dash "-", period "." and underscore "_". They can be at ost 256 characters, and must not begin with a dash "-".

To create a custom property, just use the "module:property" syntax. This is not enforced by ZFS, but is probably the cleanest approach:

```
# zfs set poudriere:type=ports tank/test/one
# zfs set poudriere:name=my_ports_tree tank/test/one
# zfs get all tank/test/one | grep poudriere
tank/test/one  poudriere:name        my_ports_tree          local
tank/test/one  poudriere:type        ports                  local
```

I am not aware of a way to remove user properties from a ZFS filesystem. As such, if it bothers you, and is cluttering up your property list, the only way to remove the user property is to create another dataset with the properties you want, copy over the data, then destroy the old cluttered dataset. Of course, you can inherit user properties with "zfs inherit" as well. And all the standard utilities, such as "zfs set", "zfs get", "zfs list", et cetera will work with user properties.

With that said, let's get to the native properties.

Native ZFS Dataset Properties

- aclinherit: Controls how ACL entries are inherited when files and directories are created. Currently, ACLs are not functioning in ZFS on Linux as of 0.6.0-rc13. Default is "restricted". Valid values for this property are:
  - discard: do not inherit any ACL properties
  - noallow: only inherit ACL entries that specify "deny" permissions
  - restricted: remove the "write_acl" and "write_owner" permissions when the ACL entry is inherited
  - passthrough: inherit all inheritable ACL entries without any modifications made to the ACL entries when they are inherited
  - passthrough-x: has the same meaning as passthrough, except that the owner@, group@, and everyone@ ACEs inherit the execute permission only if the file creation mode also requests the execute bit.
- aclmode: Controls how the ACL is modified using the "chmod" command. The value "groupmask" is default, which reduces user or group permissions. The permissions are reduced, such that they are no greater than the group permission bits, unless it is a user entry that has the same UID as the owner of the file or directory. Valid values are "discard", "groupmask", and "passthrough".
- acltype: Controls whether ACLs are enabled and if so what type of ACL to use. When a file system has the acltype property set to noacl (the default) then ACLs are disabled. Setting the acltype property to posixacl indicates Posix ACLs should be used. Posix

ACLs are specific to Linux and are not functional on other platforms. Posix ACLs are stored as an xattr and therefore will not overwrite any existing ZFS/NFSv4 ACLs which may be set. Currently only posixacls are supported on Linux.

- atime: Controls whether or not the access time of files is updated when the file is read. Default is "on". Valid values are "on" and "off".
- available: Read-only property displaying the available space to that dataset and all of its children, assuming no other activity on the pool. Can be referenced by its shortened name "avail". Availability can be limited by a number of factors, including physical space in the storage pool, quotas, reservations and other datasets in the pool.
- canmount: Controls whether the filesystem is able to be mounted when using the "zfs mount" command. Default is "on". Valid values can be "on", "off", or "noauto". When the noauto option is set, a dataset can only be mounted and unmounted explicitly. The dataset is not mounted automatically when the dataset is created or imported, nor is it mounted by the "zfs mount" command or unmounted with the "zfs unmount" command. This property is not inherited.
- casesensitivity: Indicates whether the file name matching algorithm used by the file system should be case-sensitive, case-insensitive, or allow a combination of both styles of matching. Default value is "sensitive". Valid values are "sensitive", "insensitive", and "mixed". Using the "mixed" value would be beneficial in heterogenous environments where Unix POSIX and CIFS filenames are deployed. Can only be set during dataset creation.
- checksum: Controls the checksum used to verify data integrity. The default value is "on", which automatically selects an appropriate algorithm. Currently, that algorithm is

"fletcher2". Valid values is "on", "off", "fletcher2", "fletcher4", or "sha256". Changing this property will only affect newly written data, and will not apply retroactively.

- clones: Read-only property for snapshot datasets. Displays in a comma-separated list datasets which are clones of this snapshot. If this property is not empty, then this snapshot cannot be destroyed (not even with the "-r" or "-f" options). Destroy the clone first.
- compression: Controls the compression algorithm for this dataset. Default is "off". Valid values are "on", "off", "lzjb", "gzip", "gzip-N", and "zle". The "lzjb" algorithm is optimized for speed, while provide good compression ratios. The setting of "on" defaults to "lzjb". It is recommended that you use "lzjb", "gzip", "gzip-N", or "zle" rather than "on", as the ZFS developers or package maintainers may change the algorithm "on" uses. The gzip compression algorithm uses the same compression as the "gzip" command. You can specify the gzip level by using "gzip-N" where "N" is a valid number of 1 through 9. "zle" compresses runs of binary zeroes, and is very fast. Changing this property will only affect newly written data, and will not apply retroactively.
- compressratio: Read-only property that displays the compression ratio achieved by the compression algorithm set on the "compression" property. Expressed as a multiplier. Does not take into account snapshots; see "refcompressratio". Compression is not enabled by default.
- copies: Controls the number of copies to store in this dataset. Default value is "1". Valid values are "1", "2", and "3". These copies are in addition to any redundancy provided by the pool. The copies are stored on different disks, if possible. The space used by

multiple copies is charged to the associated file and dataset. Changing this property only affects newly written data, and does not apply retroactively.

- creation: Read-only property that displays the time the dataset was created.
- defer_destroy: Read-only property for snapshots. This property is "on" if the snapshot has been marked for deferred destruction by using the "zfs destroy -d" command. Otherwise, the property is "off".
- dedup: Controls whether or not data deduplication is in effect for this dataset. Default is "off". Valid values are "off", "on", "verify", and "sha256[,verify]". The default checksum used for deduplication is SHA256, which is subject to change. When the "dedup" property is enabled, it overrides the "checksum" property. If the property is set to "verify", then if two blocks have the same checksum, ZFS will do a byte-by-byte comparison with the existing block to ensure the blocks are identical. Changing this property only affects newly written data, and is not applied retroactively. Enabling deduplication in the dataset will dedupe data in that dataset against all data in the storage pool. Disabling this property does not destroy the deduplication table. Data will continue to remain deduped.
- devices: Controls whether device nodes can be opened on this file system. The default value is "on". Valid values are "on" and "off".
- exec: Controls whether processes can be executed from within this file system. The default value is "on". Valid values are "on" and "off".
- groupquota@<group>: Limits the amount of space consumed by the specified group. Group space consumption is identified by the "userquota@<user>" property. Default value is "none". Valid values are "none", and a size in bytes.

- groupsused@<group>: Read-only property displaying the amount of space consumed by the specified group in this dataset. Space is charged to the group of each file, as displayed by "ls -l". See the userused@<user> property for more information.
- logbias: Controls how to use the SLOG, if one exists. Provides a hint to ZFS on how to handle synchronous requests. Default value is "latency", which will use a SLOG in the pool if present. The other valid value is "throughput" which will not use the SLOG on synchronous requests, and go straight to platter disk.
- mlslabel: The mlslabel property is a sensitivity label that determines if a dataset can be mounted in a zone on a system with Trusted Extensions enabled. Default value is "none". Valid values are a Solaris Zones label or "none". Note, Zones are a Solaris feature, and not relevant to GNU/Linux. However, this may be something that could be implemented with SELinux an Linux containers in the future.
- mounted: Read-only property that indicates whether the dataset is mounted. This property will display either "yes" or "no".
- mountpoint: Controls the mount point used for this file system. Default value is "<pool>/<dataset>". Valid values are an absolute path on the filesystem, "none", or "legacy". When the "mountpoint" property is changed, the new destination must not contain any child files. The dataset will be unmounted and re-mounted to the new destination.
- nbmand: Controls whether the file system should be mounted with non-blocking mandatory locks. This is used for CIFS clients. Default value is "on". Valid values are "on" and "off". Changing the property will only take effect after the dataset has ben unmounted then re-mounted.

- normalization: Indicates whether the file system should perform a unicode normalization of file names whenever two file names are compared and which normalization algorithm should be used. Default value is "none". Valid values are "formC", "formD", "formKC", and "formKD". This property cannot be changed after the dataset is created.
- origin: Read-only property for clones or volumes, which displays the snapshot from whence the clone was created.
- primarycache: Controls what is cached in the primary cache (ARC). If this property is set to "all", then both user data and metadata is cached. If set to "none", then neither are cached. If set to "metadata", then only metadata is cached. Default is "all".
- quota: Limits the amount of space a dataset and its descendents can consume. This property enforces a hard limit on the amount of space used. There is no soft limit. This includes all space consumed by descendents, including file systems and snapshots. Setting a quota on a descendant of a dataset that already has a quota does not override the ancestor's quota, but rather imposes an additional limit. Quotas cannot be set on volumes, as the volsize property acts as an implicit quota. Default value is "none" Valid values are a size in bytes or "none".
- readonly: Controls whether this dataset can be modified. The default value is off. Valid values are "on" and "off". This property can also be referred to by its shortened column name, "rdonly".
- recordsize: Specifies a suggested block size for files in the file system. This property is designed solely for use with database workloads that access files in fixed-size records. ZFS automatically tunes block sizes according to internal algorithms optimized for typical access patterns. The size specified must be a power of two greater than or equal

to 512 and less than or equal to 128 KB. Changing the file system's recordsize affects only files created afterward; existing files are unaffected. This property can also be referred to by its shortened column name, "recsize".

- refcompressratio: Read only property displaying the compression ratio achieved by the space occupied in the "referenced" property.
- referenced: Read-only property displaying the amount of data that the dataset can access. Initially, this will be the same number as the "used" property. As snapshots are created, and data is modified however, those numbers will diverge. This property can be reference by its shortened name "refer".
- refquota: Limits the amount of space a dataset can consume. This property enforces a hard limit on the amount of space used. This hard limit does not include space used by descendents, including file systems and snapshots. Default value is "none". Valid values are "none", and a size in bytes.
- refreservation: The minimum amount of space guaranteed to a dataset, not including its descendents. When the amount of space used is below this value, the dataset is treated as if it were taking up the amount of space specified by refreservation. Default value is "none". Valid values are "none" and a size in bytes. This property can also be referred to by its shortened column name, "refreserv".
- reservation: The minimum amount of space guaranteed to a dataset and its descendents. When the amount of space used is below this value, the dataset is treated as if it were taking up the amount of space specified by its reservation. Reservations are accounted for in the parent datasets' space used, and count against the parent datasets' quotas and reservations. This property can also be referred to by its shortened column name, reserv. Default value is "none". Valid values are "none" and a size in bytes.

- secondarycache: Controls what is cached in the secondary cache (L2ARC). If this property is set to "all", then both user data and metadata is cached. If this property is set to "none", then neither user data nor metadata is cached. If this property is set to "metadata", then only metadata is cached. The default value is "all".
- setuid: Controls whether the set-UID bit is respected for the file system. The default value is on. Valid values are "on" and "off".
- shareiscsi: Indicates whether a ZFS volume is exported as an iSCSI target. Currently, this is not implemented in ZFS on Linux, but is pending. Valid values will be "on", "off", and "type=disk". Other disk types may also be supported. Default value will be "off".
- sharenfs: Indicates whether a ZFS dataset is exported as an NFS export, and what options are used. Default value is "off". Valid values are "on", "off", and a list of valid NFS export options. If set to "on", the export can then be shared with the "zfs share" command, and unshared with the "zfs unshare" command. An NFS daemon must be running on the host before the export can be used. Debian and Ubuntu require a valid export in the /etc/exports file before the daemon will start.
- sharesmb: Indicates whether a ZFS dataset is export as a SMB share. Default value is "off". Valid values are "on" and "off". Currently, a bug exists preventing this from being used. When fixed, it will require a running Samba daemon, just like with NFS, and will be shared and unshared with the "zfs share" and "zfs unshare" commands.
- snapdir: Controls whether the ".zfs" directory is hidden or visible in the root of the file system. Default value is "hidden". Valid values are "hidden" and "visible". Even though the "hidden" value might be set, it is still possible to change directories into the ".zfs" directory, to access the shares and snapshots.

- sync: Controls the behavior of synchronous requests (e.g. fsync, O_DSYNC). Default value is "default", which is POSIX behavior to ensure all synchronous requests are written to stable storage and all devices are flushed to ensure data is not cached by device controllers. Valid values are "default", "always", and "disabled". The value of "always" causes every file system transaction to be written and flushed before its system call returns. The value of "disabled" does not honor synchronous requests, which will give the highest performance.
- type: Read-only property that displays the type of filesystem, whether it be a "dataset", "volume" or "snapshot".
- used: Read-only property that displays the amount of space consumed by this dataset and all its children. When snapshots are created, the space is initially shared between the parent dataset and its snapshot. As data is modified in the dataset, space that was previously shared becomes unique to the snapshot, and is only counted in the "used" property for that snapshot. Further, deleting snapshots can free up space unique to other snapshots.
- usedbychildren: Read-only property that displays the amount of space used by children of this dataset, which is freed if all of the children are destroyed.
- usedbydataset: Read-only property that displays the amount of space used by this dataset itself., which would then be freed if this dataset is destroyed.
- usedbyrefreservation: Read-only property that displays the amount of space used by a refreservation set on this dataset, which would be freed if the refreservation was removed.
- usedbysnapshots: Read-only property that displays the amount of space consumed by snapshots of this dataset. In other words, this is the data that is unique to the snapshots.

Note, this is not a sum of each snapshot's "used" property, as data can be shared across snapshots.

- userquota@<user>: Limits the amount of space consumed by the specified user. Similar to the "refquota" property, the userquota space calculation does not include space that is used by descendent datasets, such as snapshots and clones. Enforcement of user quotas may be delayed by several seconds. This delay means that a user might exceed their quota before the system notices that they are over quota and begins to refuse additional writes with the EDQUOT error message. This property is not available on volumes, on file systems before version 4, or on pools before version 15. Default value is "none". Valid values are "none" and a size in bytes.
- userrefs: Read-only property on snapshots that displays the number of user holds on this snapshot. User holds are set by using the zfs hold command.
- userused@<user>: Read-only property that displays the amount of space consumed by the specified user in this dataset. Space is charged to the owner of each file, as displayed by "ls -l". The amount of space charged is displayed by du and ls -s. See the zfs userspace subcommand for more information. The "userused@<user>" properties are not displayed with "zfs get all". The user's name must be appended after the @ symbol, using one of the following forms:
  - POSIX name (for example, joe)
  - POSIX numeric ID (for example, 789)
  - SID name (for example, joe.smith@mydomain)
  - SID numeric ID (for example, S-1-123-456-789)
- utf8only: Indicates whether the file system should reject file names that include characters that are not present in the UTF-8 character set. Default value is "off". Valid

values are "on" and "off". This property cannot be changed after the dataset has been created.

- version: The on-disk version of this file system, which is independent of the pool version. This property can only be set to later supported versions. Valid values are "current", "1", "2", "3", "4", or "5".
- volblocksize: Read-only property for volumes that specifies the block size of the volume. The blocksize cannot be changed once the volume has been written, so it should be set at volume creation time. The default blocksize for volumes is 8 KB. Any power of 2 from 512 bytes to 128 KB is valid.
- vscan: Controls whether regular files should be scanned for viruses when a file is opened and closed. In addition to enabling this property, the virus scan service must also be enabled for virus scanning to occur. The default value is "off". Valid values are "on" and "off".
- written: Read-only property that displays the amount of referenced space written to this dataset since the previous snapshot.
- written@<snapshot>: Read-only property on a snapshot that displays the amount of referenced space written to this dataset since the specified snapshot. This is the space that is referenced by this dataset but was not referenced by the specified snapshot.
- xattr: Controls whether extended attributes are enabled for this file system. The default value is "on". Valid values are "on" and "off".
- zoned: Controls whether the dataset is managed from a non-global zone. Zones are a Solaris feature and are not relevant on Linux. Default value is "off". Valid values are "on" and "off".

Final Thoughts

As you have probably noticed, some ZFS dataset properties are not fully implemented with ZFS on Linux, such as sharing a volume via iSCSI. Other dataset properties apply to the whole pool, such as the case with deduplication, even though they are applied to specific datasets. Many properties only apply to newly written data, and are not retroactive. As such, be aware of each property, and the pros/cons of what it provides. Because the parent storage pool is also a valid ZFS dataset, any child datasets will inherit non-default properties, as seen. And, the same is true for nested datasets, snapshots and volumes.

With ZFS dataset properties, you now have all the tuning at your fingertips to setup a solid ZFS storage backend. And everything has been handled with the "zfs" command, and its necessary subcommands. In fact, up to this point, we've only learned two commands: "zpool" and "zfs", yet we've been able to build and configure powerful, large, redundant, consistent, fast and tuned ZFS filesystems. This is unprecedented in the storage world, especially with GNU/Linux. The only thing left to discuss is some best practices and caveats, and then a brief post on the "zdb" command (which you should never need), and we'll be done with this series. Hell, if you've made it this far, I commend you. This has been no small series (believe me, my fingers hate me).

Posted by Aaron Toponce on Wednesday, January 2,

2013, at 6:00 am. Filed under Debian, Linux, Ubuntu,