NAME

zfs-send - generate backup stream of ZFS dataset

SYNOPSIS

zfs send [**-DLPVbcehnpsvw**] [**-R** [**-X** *dataset*[,*dataset*]<?>]] [[**-I**|**-i**] *snapshot*] *snapshot*
zfs send [**-DLPVcensvw**] [**-i** *snapshot*|*bookmark*] *filesystem*|*volume*|*snapshot*
zfs send --redact *redaction_bookmark* [**-DLPVcenpv**] [**-i** *snapshot*|*bookmark*] *snapshot*
zfs send [**-PVenv**] **-t** *receive_resume_token*
zfs send [**-PVnv**] **-S** *filesystem*
zfs redact *snapshot redaction_bookmark redaction_snapshot*<?>

DESCRIPTION

zfs send [**-DLPVbcehnpsvw**] [**-R** [**-X** *dataset*[,*dataset*]<?>]] [[**-I**|**-i**] *snapshot*] *snapshot*

Creates a stream representation of the second *snapshot*, which is written to standard output. The output can be redirected to a file or to a different system (for example, using ssh(1)). By default, a full stream is generated.

-D, **--dedup**

Deduplicated send is no longer supported. This flag is accepted for backwards compatibility, but a regular, non-deduplicated stream will be generated.

-I *snapshot*

Generate a stream package that sends all intermediary snapshots from the first snapshot to the second snapshot. For example, **-I** *@a fs@d* is similar to **-i** *@a fs@b*; **-i** *@b fs@c*; **-i** *@c fs@d*. The incremental source may be specified as with the **-i** option.

-L, **--large-block**

Generate a stream which may contain blocks larger than 128 KiB. This flag has no effect if the **large_blocks** pool feature is disabled, or if the **recordsize** property of this filesystem has never been set above 128 KiB. The receiving system must have the **large_blocks** pool feature enabled as well. See zpool-features(7) for details on ZFS feature flags and the **large_blocks** feature.

-P, **--parsable**

Print machine-parsable verbose information about the stream package generated.

-R, **--replicate**

Generate a replication stream package, which will replicate the specified file system, and all descendent file systems, up to the named snapshot. When received, all properties, snapshots, descendent file systems, and clones are preserved.

If the **-i** or **-I** flags are used in conjunction with the **-R** flag, an incremental replication stream is generated. The current values of properties, and current snapshot and file system names are set when the stream is received. If the **-F** flag is specified when this stream is received, snapshots and file systems that do not exist on the sending side are destroyed. If the **-R** flag is used to send encrypted datasets, then **-w** must also be specified.

-V, **--proctitle**
Set the process title to a per-second report of how much data has been sent.

-X, **--exclude** *dataset*[,*dataset*]<?>
With **-R**, **-X** specifies a set of datasets (and, hence, their descendants), to be excluded from the send stream. The root dataset may not be excluded. **-X** *a* **-X** *b* is equivalent to **-X** *a*,*b*.

-e, **--embed**
Generate a more compact stream by using **WRITE_EMBEDDED** records for blocks which are stored more compactly on disk by the **embedded_data** pool feature. This flag has no effect if the **embedded_data** feature is disabled. The receiving system must have the **embedded_data** feature enabled. If the **lz4_compress** feature is active on the sending system, then the receiving system must have that feature enabled as well. Datasets that are sent with this flag may not be received as an encrypted dataset, since encrypted datasets cannot use the **embedded_data** feature. See zpool-features(7) for details on ZFS feature flags and the **embedded_data** feature.

-b, **--backup**
Sends only received property values whether or not they are overridden by local settings, but only if the dataset has ever been received. Use this option when you want **zfs receive** to restore received properties backed up on the sent dataset and to avoid sending local settings that may have nothing to do with the source dataset, but only with how the data is backed up.

-c, **--compressed**
Generate a more compact stream by using compressed WRITE records for blocks which are compressed on disk and in memory (see the **compression** property for details). If the **lz4_compress** feature is active on the sending system, then the receiving system must have that feature enabled as well. If the **large_blocks** feature is enabled on the sending system but the **-L** option is not supplied in conjunction with **-c**, then the data will be decompressed before sending so it can be split into smaller block sizes. Streams sent with **-c** will not have their data recompressed on the receiver side using **-o compress**= *value*. The data will stay compressed as it was from the sender. The new compression property will be set for future data. Note that uncompressed data from the sender will still attempt to compress on the receiver, unless you specify **-o compress**= *off*.

-w, **--raw**

For encrypted datasets, send data exactly as it exists on disk. This allows backups to be taken even if encryption keys are not currently loaded. The backup may then be received on an untrusted machine since that machine will not have the encryption keys to read the protected data or alter it without being detected. Upon being received, the dataset will have the same encryption keys as it did on the send side, although the **keylocation** property will be defaulted to **prompt** if not otherwise provided. For unencrypted datasets, this flag will be equivalent to **-Lec**. Note that if you do not use this flag for sending encrypted datasets, data will be sent unencrypted and may be re-encrypted with a different encryption key on the receiving system, which will disable the ability to do a raw send to that system for incrementals.

-h, **--holds**
Generate a stream package that includes any snapshot holds (created with the **zfs hold** command), and indicating to **zfs receive** that the holds be applied to the dataset on the receiving system.

-i *snapshot*
Generate an incremental stream from the first *snapshot* (the incremental source) to the second *snapshot* (the incremental target). The incremental source can be specified as the last component of the snapshot name (the **@** character and following) and it is assumed to be from the same file system as the incremental target.

If the destination is a clone, the source may be the origin snapshot, which must be fully specified (for example, *pool/fs@origin*, not just *@origin*).

-n, **--dryrun**
Do a dry-run ("No-op") send. Do not generate any actual send data. This is useful in conjunction with the **-v** or **-P** flags to determine what data will be sent. In this case, the verbose output will be written to standard output (contrast with a non-dry-run, where the stream is written to standard output and the verbose output goes to standard error).

-p, **--props**
Include the dataset's properties in the stream. This flag is implicit when **-R** is specified. The receiving system must also support this feature. Sends of encrypted datasets must use **-w** when using this flag.

-s, **--skip-missing**
Allows sending a replication stream even when there are snapshots missing in the hierarchy. When a snapshot is missing, instead of throwing an error and aborting the send, a warning is printed to the standard error stream and the dataset to which it belongs and its descendents are skipped. This flag can only be used in conjunction with **-R**.

-v, **--verbose**
Print verbose information about the stream package generated. This information includes a per-second report of how much data has been sent. The same report can be requested by sending SIGINFO or SIGUSR1, regardless of **-v**.

The format of the stream is committed. You will be able to receive your streams on future versions of ZFS.

zfs send [**-DLPVcenvw**] [**-i** *snapshot*|*bookmark*] *filesystem*|*volume*|*snapshot*
Generate a send stream, which may be of a filesystem, and may be incremental from a bookmark. If the destination is a filesystem or volume, the pool must be read-only, or the filesystem must not be mounted. When the stream generated from a filesystem or volume is received, the default snapshot name will be "--head--".

-D, **--dedup**
Deduplicated send is no longer supported. This flag is accepted for backwards compatibility, but a regular, non-deduplicated stream will be generated.

-L, **--large-block**
Generate a stream which may contain blocks larger than 128 KiB. This flag has no effect if the **large_blocks** pool feature is disabled, or if the **recordsize** property of this filesystem has never been set above 128 KiB. The receiving system must have the **large_blocks** pool feature enabled as well. See zpool-features(7) for details on ZFS feature flags and the **large_blocks** feature.

-P, **--parsable**
Print machine-parsable verbose information about the stream package generated.

-c, **--compressed**
Generate a more compact stream by using compressed WRITE records for blocks which are compressed on disk and in memory (see the **compression** property for details). If the **lz4_compress** feature is active on the sending system, then the receiving system must have that feature enabled as well. If the **large_blocks** feature is enabled on the sending system but the **-L** option is not supplied in conjunction with **-c**, then the data will be decompressed before sending so it can be split into smaller block sizes.

-w, **--raw**
For encrypted datasets, send data exactly as it exists on disk. This allows backups to be taken even if encryption keys are not currently loaded. The backup may then be received on an untrusted machine since that machine will not have the encryption keys to read the protected data or alter it without being detected. Upon being received, the dataset will have the same encryption keys as it

did on the send side, although the **keylocation** property will be defaulted to **prompt** if not otherwise provided. For unencrypted datasets, this flag will be equivalent to **-Lec**. Note that if you do not use this flag for sending encrypted datasets, data will be sent unencrypted and may be re-encrypted with a different encryption key on the receiving system, which will disable the ability to do a raw send to that system for incrementals.

-e, **--embed**
Generate a more compact stream by using **WRITE_EMBEDDED** records for blocks which are stored more compactly on disk by the **embedded_data** pool feature. This flag has no effect if the **embedded_data** feature is disabled. The receiving system must have the **embedded_data** feature enabled. If the **lz4_compress** feature is active on the sending system, then the receiving system must have that feature enabled as well. Datasets that are sent with this flag may not be received as an encrypted dataset, since encrypted datasets cannot use the **embedded_data** feature. See zpool-features(7) for details on ZFS feature flags and the **embedded_data** feature.

-i *snapshot|bookmark*
Generate an incremental send stream. The incremental source must be an earlier snapshot in the destination's history. It will commonly be an earlier snapshot in the destination's file system, in which case it can be specified as the last component of the name (the **#** or **@** character and following).

If the incremental target is a clone, the incremental source can be the origin snapshot, or an earlier snapshot in the origin's filesystem, or the origin's origin, etc.

-n, **--dryrun**
Do a dry-run ("No-op") send. Do not generate any actual send data. This is useful in conjunction with the **-v** or **-P** flags to determine what data will be sent. In this case, the verbose output will be written to standard output (contrast with a non-dry-run, where the stream is written to standard output and the verbose output goes to standard error).

-v, **--verbose**
Print verbose information about the stream package generated. This information includes a per-second report of how much data has been sent. The same report can be requested by sending SIGINFO or SIGUSR1, regardless of **-v**.

zfs send --redact *redaction_bookmark* [**-DLPVcenpv**] [**-i** *snapshot|bookmark*] *snapshot*
Generate a redacted send stream. This send stream contains all blocks from the snapshot being sent that aren't included in the redaction list contained in the bookmark specified by the **--redact** (or **-d**) flag. The resulting send stream is said to be redacted with respect to the snapshots the bookmark specified by the **--redact** flag was created with. The bookmark must have been created by running **zfs redact** on

the snapshot being sent.

This feature can be used to allow clones of a filesystem to be made available on a remote system, in the case where their parent need not (or needs to not) be usable. For example, if a filesystem contains sensitive data, and it has clones where that sensitive data has been secured or replaced with dummy data, redacted sends can be used to replicate the secured data without replicating the original sensitive data, while still sharing all possible blocks. A snapshot that has been redacted with respect to a set of snapshots will contain all blocks referenced by at least one snapshot in the set, but will contain none of the blocks referenced by none of the snapshots in the set. In other words, if all snapshots in the set have modified a given block in the parent, that block will not be sent; but if one or more snapshots have not modified a block in the parent, they will still reference the parent's block, so that block will be sent. Note that only user data will be redacted.

When the redacted send stream is received, we will generate a redacted snapshot. Due to the nature of redaction, a redacted dataset can only be used in the following ways:

1. To receive, as a clone, an incremental send from the original snapshot to one of the snapshots it was redacted with respect to. In this case, the stream will produce a valid dataset when received because all blocks that were redacted in the parent are guaranteed to be present in the child's send stream. This use case will produce a normal snapshot, which can be used just like other snapshots.

2. To receive an incremental send from the original snapshot to something redacted with respect to a subset of the set of snapshots the initial snapshot was redacted with respect to. In this case, each block that was redacted in the original is still redacted (redacting with respect to additional snapshots causes less data to be redacted (because the snapshots define what is permitted, and everything else is redacted)). This use case will produce a new redacted snapshot.

3. To receive an incremental send from a redaction bookmark of the original snapshot that was created when redacting with respect to a subset of the set of snapshots the initial snapshot was created with respect to anything else. A send stream from such a redaction bookmark will contain all of the blocks necessary to fill in any redacted data, should it be needed, because the sending system is aware of what blocks were originally redacted. This will either produce a normal snapshot or a redacted one, depending on whether the new send stream is redacted.

4. To receive an incremental send from a redacted version of the initial snapshot that is redacted with respect to a subject of the set of snapshots the initial snapshot was created with respect to. A send stream from a compatible redacted dataset will contain all of the blocks necessary to fill in any redacted data. This will either produce a normal snapshot or a redacted one, depending on whether the new send stream is redacted.

5. To receive a full send as a clone of the redacted snapshot. Since the stream is a full send, it definitionally contains all the data needed to create a new dataset. This use case will either produce a normal snapshot or a redacted one, depending on whether the full send stream was redacted.

These restrictions are detected and enforced by **zfs receive**; a redacted send stream will contain the list of snapshots that the stream is redacted with respect to. These are stored with the redacted snapshot, and are used to detect and correctly handle the cases above. Note that for technical reasons, raw sends and redacted sends cannot be combined at this time.

zfs send [**-PVenv**] **-t** *receive_resume_token*
Creates a send stream which resumes an interrupted receive. The *receive_resume_token* is the value of this property on the filesystem or volume that was being received into. See the documentation for **zfs receive -s** for more details.

zfs send [**-PVnv**] [**-i** *snapshot*|*bookmark*] **-S** *filesystem*
Generate a send stream from a dataset that has been partially received.

-S, **--saved**
This flag requires that the specified filesystem previously received a resumable send that did not finish and was interrupted. In such scenarios this flag enables the user to send this partially received state. Using this flag will always use the last fully received snapshot as the incremental source if it exists.

zfs redact *snapshot redaction_bookmark redaction_snapshot*<?>
Generate a new redaction bookmark. In addition to the typical bookmark information, a redaction bookmark contains the list of redacted blocks and the list of redaction snapshots specified. The redacted blocks are blocks in the snapshot which are not referenced by any of the redaction snapshots. These blocks are found by iterating over the metadata in each redaction snapshot to determine what has been changed since the target snapshot. Redaction is designed to support redacted zfs sends; see the entry for **zfs send** for more information on the purpose of this operation. If a redact operation fails partway through (due to an error or a system failure), the redaction can be resumed by rerunning the same command.

Redaction
ZFS has support for a limited version of data subsetting, in the form of redaction. Using the **zfs redact** command, a **redaction bookmark** can be created that stores a list of blocks containing sensitive information. When provided to **zfs send**, this causes a **redacted send** to occur. Redacted sends omit the blocks containing sensitive information, replacing them with REDACT records. When these send streams are received, a **redacted dataset** is created. A redacted dataset cannot be mounted by default, since it is incomplete. It can be used to receive other send streams. In this way datasets can be used for

data backup and replication, with all the benefits that zfs send and receive have to offer, while protecting sensitive information from being stored on less-trusted machines or services.

For the purposes of redaction, there are two steps to the process. A redact step, and a send/receive step. First, a redaction bookmark is created. This is done by providing the **zfs redact** command with a parent snapshot, a bookmark to be created, and a number of redaction snapshots. These redaction snapshots must be descendants of the parent snapshot, and they should modify data that is considered sensitive in some way. Any blocks of data modified by all of the redaction snapshots will be listed in the redaction bookmark, because it represents the truly sensitive information. When it comes to the send step, the send process will not send the blocks listed in the redaction bookmark, instead replacing them with REDACT records. When received on the target system, this will create a redacted dataset, missing the data that corresponds to the blocks in the redaction bookmark on the sending system. The incremental send streams from the original parent to the redaction snapshots can then also be received on the target system, and this will produce a complete snapshot that can be used normally. Incrementals from one snapshot on the parent filesystem and another can also be done by sending from the redaction bookmark, rather than the snapshots themselves.

In order to make the purpose of the feature more clear, an example is provided. Consider a zfs filesystem containing four files. These files represent information for an online shopping service. One file contains a list of usernames and passwords, another contains purchase histories, a third contains click tracking data, and a fourth contains user preferences. The owner of this data wants to make it available for their development teams to test against, and their market research teams to do analysis on. The development teams need information about user preferences and the click tracking data, while the market research teams need information about purchase histories and user preferences. Neither needs access to the usernames and passwords. However, because all of this data is stored in one ZFS filesystem, it must all be sent and received together. In addition, the owner of the data wants to take advantage of features like compression, checksumming, and snapshots, so they do want to continue to use ZFS to store and transmit their data. Redaction can help them do so. First, they would make two clones of a snapshot of the data on the source. In one clone, they create the setup they want their market research team to see; they delete the usernames and passwords file, and overwrite the click tracking data with dummy information. In another, they create the setup they want the development teams to see, by replacing the passwords with fake information and replacing the purchase histories with randomly generated ones. They would then create a redaction bookmark on the parent snapshot, using snapshots on the two clones as redaction snapshots. The parent can then be sent, redacted, to the target server where the research and development teams have access. Finally, incremental sends from the parent snapshot to each of the clones can be sent to and received on the target server; these snapshots are identical to the ones on the source, and are ready to be used, while the parent snapshot on the target contains none of the username and password data present on the source, because it was removed by the redacted send operation.

SIGNALS

See **-v**.

EXAMPLES

Example 1: Remotely Replicating ZFS Data

The following commands send a full stream and then an incremental stream to a remote machine, restoring them into *poolB/received/fs@a* and *poolB/received/fs@b*, respectively. *poolB* must contain the file system *poolB/received*, and must not initially contain *poolB/received/fs*.

```
# zfs send pool/fs@a |
    ssh host zfs receive poolB/received/fs@a
# zfs send -i a pool/fs@b |
    ssh host zfs receive poolB/received/fs
```

Example 2: Using the **zfs receive -d** Option

The following command sends a full stream of *poolA/fsA/fsB@snap* to a remote machine, receiving it into *poolB/received/fsA/fsB@snap*. The *fsA/fsB@snap* portion of the received snapshot's name is determined from the name of the sent snapshot. *poolB* must contain the file system *poolB/received*. If *poolB/received/fsA* does not exist, it is created as an empty file system.

```
# zfs send poolA/fsA/fsB@snap |
    ssh host zfs receive -d poolB/received
```

SEE ALSO

zfs-bookmark(8), zfs-receive(8), zfs-redact(8), zfs-snapshot(8)