ZFS Linux Developers Reference

ZFS Linux Developers Reference

Table of Contents

Chapter 1. Getting Started

Maintaining this document

This document is in the git repository at git://github.com/zfs-linux/zfs-web.git. It is a docbook xml document and is maintained by the developer community. This section will cover how you can build this document and expand on this document by adding to it. Extending this document does not need you to be a expert on xml with a few minutes of reading the crash course and setting up the tool chain you should be good to go.

Tool Chain

In the root dir of the repository there is a script call create.sh. It takes an .xml file as a parameter and compiles are create two files.

A monolithic html file
A pdf document
The first time you try this it is mostly going to fail since the tools required to install these are not present. I did not make a note of the packages needed to install. You will have to search and install the packages until the script runs successfully. Anybody who tries this please update the document so that other can just install the required packages.

Docbook xml crash course

To get you started there is a short guide in the repository located at docs/docbook-crash-course.pdf This is a good reference for the various xml tags commonly used. If you are an emacs user using the nxml-mode in emacs gives automatic validation of the tags, completion on the tags and many other useful features. I would urge other to add the other list of tools/editors/ides they have found useful.

Getting and Building from source

The ZFS on linux functionality is provided by three modules which are maintained in seperate source trees. These are

spl (solaris porting layer)
zfs (core dmu/dsl functionality)
lzfs (linux posix layer)
You can build from source in two ways :

1) Get the script from https://github.com/zfs-linux/misc-scripts/blob/master/install.sh and run it.

This will automate the whole build process.

2) You can follow the procedure below.

you need to retrive the sources for all three and compile them. If any one of them are missing zfs won't function. The three repositories can be accessed at the following url https://github.com/zfs-linux

The commands and procedures required to build fresh modules from source are listed below. Please note that some of the tools used in the procedure might not be installed on your machine and the error that result don't always clearly indicate that the package was missing.

```
/tmp$  git clone git://github.com/zfs-linux/spl.git
Initialized empty Git repository in /tmp/spl/.git/
remote: Counting objects: 4266, done.
remote: Compressing objects: 100% (1144/1144), done.
remote: Total 4266 (delta 3155), reused 4162 (delta 3078)
Receiving objects: 100% (4266/4266), 1.70 MiB | 123 KiB/s, done.
Resolving deltas: 100% (3155/3155), done.

/tmp$ git clone git://github.com/zfs-linux/zfs.git
Initialized empty Git repository in /tmp/zfs/.git/
remote: Counting objects: 68496, done.
remote: Compressing objects:   3% (631/21029)
....

/tmp$ git clone git://github.com/zfs-linux/lzfs.git
Initialized empty Git repository in /tmp/lzfs/.git/
remote: Counting objects: 173, done.
remote: Compressing objects: 100% (152/152), done.
remote: Total 173 (delta 92), reused 38 (delta 16)
Receiving objects: 100% (173/173), 199.19 KiB | 103 KiB/s, done.
Resolving deltas: 100% (92/92), done.

/tmp$ cd spl
/tmp/spl$ ./configure --with-linux=/lib/modules/2.6.32-24-server/build
checking metadata... yes
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking whether to enable maintainer-specific portions of Makefiles... no
checking for a BSD-compatible install... /usr/bin/install -c
....

/tmp/spl$ make
make  all-recursive
make[1]: Entering directory `/tmp/spl'
Making all in lib
make[2]: Entering directory `/tmp/spl/lib'
/bin/bash ../libtool  --tag=CC --silent  --mode=compile gcc -DHAVE_CONFIG_H -inclu
../spl_config.h    -Wall -Wshadow -Wstrict-prototypes -fno-strict-aliasing
-D__USE_LARGEFILE64 -DNDEBUG -g -O
....

/tmp/spl$ cd ../zfs/
/tmp/zfs$ ./configure --with-linux=/lib/modules/2.6.32-24-server/build --with-spl=
checking metadata... yes
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking whether to enable maintainer-specific portions of Makefiles... no
....
/tmp/zfs$ make
make  all-recursive
make[1]: Entering directory `/tmp/zfs'
```

```
Making all in etc
make[2]: Entering directory `/tmp/zfs/etc'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/tmp/zfs/etc'
....
/tmp/zfs$ cd ../lzfs/
/tmp/lzfs$ ./configure --with-linux=/lib/modules/2.6.32-24-server/build --with-spl
checking metadata... yes
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
....
/tmp/lzfs$ make
make  all-recursive
make[1]: Entering directory `/tmp/lzfs'
Making all in module
make[2]: Entering directory `/tmp/lzfs/module'
make -C /lib/modules/2.6.32-24-server/build SUBDIRS=`pwd` V=1 modules
....
/tmp/lzfs$ cd ../zfs/scripts/
/tmp/zfs/scripts$ ./zfs.sh -v
Loading zlib_deflate (/lib/modules/2.6.32-24-server/kernel/lib/
zlib_deflate/zlib_deflate.ko)
Loading spl (/tmp/spl//module/spl/spl.ko)
Loading splat (/tmp/spl//module/splat/splat.ko)
Loading zavl (/tmp/zfs/module/avl/zavl.ko)
Loading znvpair (/tmp/zfs/module/nvpair/znvpair.ko)
....
/tmp/zfs/scripts$ insmod /tmp/lzfs/module/lzfs.ko
/tmp/zfs/scripts$ cd /tmp/spl/
/tmp/spl$ make install
Making install in lib
make[1]: Entering directory `/tmp/spl/lib'
make[2]: Entering directory `/tmp/spl/lib'
make[2]: Nothing to be done for `install-exec-am'.
make[2]: Nothing to be done for `install-data-am'.
....
/tmp/spl$ cd ../zfs/
/tmp/zfs$ make install
Making install in etc
make[1]: Entering directory `/tmp/zfs/etc'
make[2]: Entering directory `/tmp/zfs/etc'
test -z "/etc" || /bin/mkdir -p "/etc"
/bin/mkdir -p '/etc/../etc/udev/rules.d'
....
/tmp/zfs$ cd ../lzfs/
/tmp/lzfs$ make install
Making install in module
make[1]: Entering directory `/tmp/lzfs/module'
make -C /lib/modules/2.6.32-24-server/build SUBDIRS=`pwd` \
  INSTALL_MOD_PATH= \
  INSTALL_MOD_DIR=addon/lzfs modules_install
```

```
....
/tmp/lzfs$ lsmod | grep lzfs
lzfs                    28371  0
zfs                    964150  1 lzfs
spl                    120247  7 lzfs,zfs,zcommon,zunicode,znvpair,zavl,splat
```

Building Debian and RPM Packages

Building Debian Packages:

Install the required packages to build debian packages :

```
$ sudo apt-get install build-essential devscripts ubuntu-dev-tools debhelper dh-ma
```

Install other required packages:

```
$ sudo apt-get install gawk uuid-dev linux-headers zlib1g-dev
```

*** Building deb package for SPL:**

Clone the repository :

```
$ git clone https://github.com/zfs-linux/spl.git
```

Debian style packaging needs the tar.gz file for the source code:

```
$ mv spl/ spl-0.5.2/
$ tar czf spl-0.5.2.tar.gz spl-0.5.2/
```

Make a back copy of the source :

```
$ cp spl-0.5.2.tar.gz spl_0.5.2.orig.tar.gz
```

```
$ cd spl-0.5.2/
```

Run following command to create debian package

```
$ sudo debuild -i -us -uc
```

Package will be created in the parent directory :

Building deb package for ZFS :

Install the earlier created SPL package, as it is needed to build ZFS package.

Clone the repository :

```
$ git clone https://github.com/zfs-linux/zfs.git
```

Debian style packaging needs the tar.gz file for the source code.

```
$ mv zfs/ zfs-0.5.1/
$ tar czf zfs-0.5.1.tar.gz zfs-0.5.1
```

Make a back copy of the source :

```
$ cp zfs-0.5.1.tar.gz zfs_0.5.1.orig.tar.gz
```

```
$ cd zfs-0.5.1
```

Run the following command to create debian package :

```
$ sudo debuild -i -us -uc
```

Package will be create in parent directory.

Building deb package for LZFS:

Install the earlier created SPL and ZFS packages, as it is needed to build LZFS package.

Clone the repository:

```
$  git clone https://github.com/zfs-linux/lzfs.git
```

Debian style packaging needs the tar.gz file for the source code.

```
$ mv lzfs/ lzfs-1.0/
```

```
$ tar czf lzfs-1.0.tar.gz lzfs-1.0
```

Make a back copy of the source.

```
$ cp lzfs-1.0.tar.gz lzfs_1.0.orig.tar.gz
```

```
$ cd lzfs-1.0
```

Run following command to create debian package :

```
$ sudo debuild -i -us -uc
```

Package will be created in parent directory.

Building RPM Packages :

Install the required packages to build RPM packages:

```
$ yum install rpm-build
```

Building RPM package for SPL

Clone the repository:

```
$ git clone https://github.com/zfs-linux/spl.git
```

Run following commands:-

```
$ cd spl
$ ./configure
```

To create the RPM run

```
$ make rpm
```

Building RPM package for ZFS

Clone the repository

```
$ git clone https://github.com/zfs-linux/zfs.git
```

```
$ cd zfs
```

Run ./configure with specific SPL source against which we want to compile the ZFS.

```
$ ./configure --withs-spl=<SPL_SOURCE_CODE>
```

Or you can install the SPL rpm and run

```
$ ./configure
```

To create the RPM run

```
$ make rpm
```

Building RPM package for LZFS:

Clone the repository

```
$ git clone https://github.com/zfs-linux/lzfs.git
```

```
$ cd lzfs
```

Run ./configure with specific SPL source and ZFS source against which we want to compile the LZFS.

```
$ ./configure --withs-spl=<SPL_SOURCE_CODE> --with-zfs=<ZFS_SOURCE_CODE>
```

Or you can install the SPL and ZFS rpms and run

```
$ ./configure
```

To create the RPM run

```
$ make rpm
```

Chapter 2. Debugging Infrastructure

printk, cmn_err and dmesg

If you have worked on this a short time you would realize that although cmn_err prints to the dmesg buffer it is not very reliable. You cannot be sure that messages have not been skipped. There is a parallel infrastructure in SPL which gives much more reliable logging.

The messages from cmn_err depending on the settings go to the dmesg buffer as well as the spl log buffer. This spl log buffer is in core. To dump this to a file you need to execute "echo 1 >/proc/sys/kernel/spl/debug/dump" This will create a binary file in /tmp/spl*. Use the spl command to cover this binary file to text.

How to debug using printk and cmn_err

Put the cmn_err statements in the path code which you want to debug

```
/tmp$  dmesg -c
/tmp$  The command by which you are expecting the cmn_err messages
/tmp$  echo 1>/proc/sys/kernel/spl/debug/dump
/tmp$  spl /tmp/spl-log.1293106961.31507
```