

ZFS Linux Developers Reference

ZFS Linux Developers Reference

Table of Contents

- 1. Getting Started 1
 - Maintaining this document 1
 - Tool Chain 1
 - Docbook xml crash course 1
 - Getting and Building from source 1
 - Installing Startup Scripts 4
- 2. Debugging Infrastructure 5
 - printk, cmn_err and dmesg 5

Chapter 1. Getting Started

Maintaining this document

This document is in the git repository at [git://github.com/zfs-linux/zfs-web.git](https://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 an 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 and creates 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 others 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 others 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 separate source trees. These are

`spl` (solaris porting layer)

`zfs` (core dmu/dsl functionality)

`lzfs` (linux posix layer)

You need to retrieve 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 results 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 -include
../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=/tmp/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=/tmp/spl/
--with-zfs=/tmp/zfs/
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
```

Installing Startup Scripts

The binaries have been installed. Currently the make system does not install the startup scripts these have to be done manually.

Follow this procedure for Fedora

```
/tmp$ chkconfig --add zfsload
```

Follow this procedure for Ubuntu

```
/tmp$ cp lzfs/scripts/zfsload-ubuntu /etc/init.d/zfsload  
/tmp$ chown root /etc/init.d/zfsload  
/tmp$ chmod +x /etc/init.d/zfsload  
/tmp$ update-rc.d zfsload defaults  
/tmp$ service zfsload start
```

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.