Linux + XFS HOWTO

Linux on Steroids

Russel Ingram

ringram@gargoylecc.com

Revision History

Revision v1.1	2002−05−12	Revised by: ri
Updated sgi cvs info to match current state; made various changes/clarifications based on reader feedback		
Revision v1.02	2001−10−08	Revised by: ri
Added some note, blockquote and ulink tags. Corrected error in command section of "Finishing Up". Changed note about e2fsprogs−devel to refer to libuuid.		
Revision v1.01	2001−10−04	Revised by: ri
Corrected error in "Finishing up" section; various formatting changes		

This document describes how to build a Linux system that runs on top of the SGI XFS journaling filesystem.

Table of Contents

1. Introduction

1.1. Introduction to XFS for Linux

This document describes how to build a Linux system that runs on top of the SGI XFS journaling filesystem. From the XFS FAQ: "XFS is a journalling filesystem developed by SGI and used in SGI's IRIX operating system. It is now also available under GPL for linux. It is extremely scalable, using btrees extensively to support large sparse files, and extremely large directories. The journalling capability means no more waiting for fsck's or worrying about meta−data corruption." Essentialy XFS is the filesystem SGI designed for its highend server systems, hence the subtitle of this document, "Linux on Steroids". :−)

1.2. Foreword, Feedback, & Credits

As a fairly new member of the Irix System Administrator community I have fallen in love with the robustness of the filesystem that has been developed to support Irix (XFS of course). So, needless to say I've been following the porting effort to Linux for quite some time and have dreamed of being able to run my Linux systems on top of an all XFS filesystem since the beginning. The project has come to a point (actually nearly a year ago at the time of this writing) that this could actually be a reallity. However, as is the case with a great deal of programming/porting projects, the documentation for such task is not always as readily available or easy to follow as one might hope. This document is an attempt to remedy that situation.

The information contained in this document is based on messages from Jason Walker and Russell Cattelan taken from the XFS development mailing list, information gathered from various places on the SGI Open Source Development web site, and from my own experiences setting up such a system.

Please feel free to email me at <ringram@gargoylecc.com> if you have any corrections or if any imformation/URLs/etc. is missing. The more feedback I get on this HOWTO the more helpful it will be for all.

The latest version of this document can be found at Gargoyle Computer Consulting .

1.3. Copyright & Disclaimer

2. Preparation for XFS Installation

2.1. Downloading the Linux 2.4.x−XFS Kernel Source

Currently the only place to get the source code for the XFS enabled Linux kernel is straight from SGI's Open Source Development site via CVS.

Note

two distinct trees are available:

- linux−2.5−xfs: development tree
- linux−2.4−xfs: stable bug fix only tree

My experience has been with the 2.4 tree, but I imagine everything will work the same with the development tree. Both trees are kept in sync with their respective mainline kernel tree at least to the point of major release numbers.

Here are the steps to download the kernel source tree:

A. Normally the linux kernel source is installed in the /usr/src directory, so you should start off by switching to that directory.

```
$ cd /usr/src
```

B. Next, you should set the CVSROOT environment variable so that it points to the proper cvs server.

- If you are running sh, bash, ksh, etc...:
  ```
  $ export CVSROOT=':pserver:cvs@oss.sgi.com:/cvs'
  ```
- If you are running csh or tcsh:
  ```
  $ setenv CVSROOT  :pserver:cvs@oss.sgi.com:/cvs
  ```

If you plan on updating your kernel often (to keep up with the latest changes) you might want to put this in your login script.

C. Then log in to the cvs server.

```
$ cvs login  (the password is "cvs")
```

This needs to be done only ONCE, not everytime you access CVS.

D. Now grab linux−2.4−xfs. The first time you will want to do something like:

```
$ cvs −z3 co linux−2.4−xfs
```

After you have checked the code out, you can use:

```
$ cvs −z3 update linux−2.4−xfs
```

to update your copy to the latest version from the CVS server.

2.2. XFS Support as Modules or Compiled Into the Kernel?

The option to build XFS support for the Linux kernel as modules is available and will work (or so I am told) with the help of an initial RAM disk and a couple of additions to the lilo configuration. I have not tried this (yet), so I will not include documentation on how this is done other than just to qoute from a message to the development mailing list from Russell Cattelan:

> Actually running xfs as a module isn't very hard. in the directory cmd/xfs/misc there is a modified mkinitrd the will always generate a ram disk with pagebuf xfs_support and xfs.
>
> Once that is done just add the initrd line in lilo.conf AND
>
> ```
> append = "ramdisk_size=25000"
> ```
>
> The default size is 4096 which isn't nearly large enough to hold xfs.
>
> This is from my laptop.
>
> ```
> punch[12:57am]−=>mount
> /dev/ide/host0/bus0/target0/lun0/part8 on / type xfs (rw,noatime)
> none on /proc type proc (rw)
> /dev/ide/host0/bus0/target0/lun0/part6 on /boot type ext2 (rw,noatime)
> none on /dev/pts type devpts (rw,mode=0620)
> /dev/ide/host0/bus0/target0/lun0/part1 on /mnt/windows type vfat (rw,nosuid,nodev,umask=0
> /dev/ide/host0/bus0/target0/lun0/part9 on /blam type xfs (rw)
>
> punch[12:57am]−=>lsmod
> Module                  Size  Used by
> autofs                 13180   1 (autoclean)
> usb−uhci               24918   0 (unused)
> usbcore                35339   0 [usb−uhci]
> 3c59x                  25149   1 (autoclean)
> maestro                29757   0 (unused)
> soundcore               6085   2 [maestro]
> vfat                   13075   1 (autoclean)
> fat                    37733   0 (autoclean) [vfat]
> xfs                   447888   2
> xfs_support            13954   0 [xfs]
> pagebuf                39935   2 [xfs]
>
>
> image=/boot/vmlinuz−2.4.0−XFS−test13−pre4
> label=t13p4
> root=/dev/hda8
> ```

```
initrd=/boot/initrd−2.4.0−XFS−test13p4.img
append="ramdisk_size=25000"
read−only
```

−− Russell Cattelan

It seems to me that compiling the support into the kernel would be much simpler, so that is how I am doing it at this point. I will try it as a module at a later time and add more detailed instructions then. If anyone has time to document this method before I get around to it please email it to me and I will include it with credit given where credit is due. :−)

3. Kernel Configuration and Installation

3.1. Configuring your kernel for XFS support

Note

If you have never configured and compiled a new linux kernel you might consider reading the Linux Kernel HOWTO before doing this step. It can be found at the Linux Documentation Project (LDP) or one of its mirrors.

After having downloaded the cvs source tree the actual kernel source will be in /usr/src/linux−2.4−xfs(−beta)/linux, so you should switch to that directory before running the make config command of your choice. The main things that must be included in your kernel to provide XFS support are "Page Buffer support" and (duh) "SGI XFS filesystem support." Both options are available in the "File systems" section of the kernel configuration. You will need to have "Prompt for development and/or incomplete code/drivers" selected under "Code maturity level options" for those options to be available to you. Optionally, you may also want to select "Enable XFS Debug mode" and "Enable XFS Vnode Tracing" under "SGI XFS filesystem support." These options may slow your XFS implementation some, but may be useful in tracing the cause of a crash if a crash occurs.

3.2. Building the kernel and modules

As with any kernel build, the following commands must be run to actually build the new kernel:

```
$ make dep
$ make bzImage
$ make modules
```

3.3. Installing the new kernel and modules

Again this is standard for any kernel installation:

```
$ make modules_install
$ cp arch/i386/boot/bzImage /boot/vmlinuz−2.4.0−XFS
```

3.4. Add a new entry to your lilo configuration and re−install lilo

```
$ vi /etc/lilo.conf
```

Add a new image section to your lilo.conf file similar to the following:

```
image=/boot/vmlinuz-2.4.0-XFS label=xfs  read-only  root=/dev/hda2
```

The "root=" line should match the "root=" line from the existing image sections in your lilo.conf file. Don't forget to run lilo when you're through editing lilo.conf to make the changes effective.

3.5. Build and install the XFS utilities

There are a number of tools that come with the XFS filesystem that allow you to build and manage your XFS filesystems that must be built as well. These tools are in the /usr/src/linux−2.4−xfs(−beta)/cmd/xfsprogs directory.

Note

These tools rely on the /usr/lib/libuuid.a shared library. If you do not have this library installed you will need it in order for the XFS utilities to compile. You can find the rpm package for your version of Linux from Rpmfind.net by searching for "/usr/lib/libuuid.a." The debian package that contains libuuid is uuid−dev. There will no doubt be other distributions that package this library in another place. A good way to find the correct package on those distributions is to search on the Google Linux search engine.

Change to that directory:

```
$ cd ../cmd/xfsprogs
```

Build and install the xfs utilities:

```
$ make install
```

3.6. Boot the new kernel

```
$ reboot
```

Note

Unless you changed the default label to boot in your lilo.conf file you will need to enter "xfs" at the "LILO Boot:" prompt in order to boot the new kernel image.

4. Filesystem Migration

The final part of this whole process is probably actually the trickiest and most dangerous as far as the possibility of losing your data goes. I strongly suggest that you make a complete backup of the system (or at least all important data) before attempting the migration to XFS. This part is also the most difficult to explain as there are probably hundreds of ways you can do this based on the set up of your existing filesystem. I will give you the basic commands for creating the new filesystems, try to give some pointers on how to go about shuffling filesystems, and in general just relay to you the way I went about migrating my own filesystems.

4.1. Migrating the / filesystem

Probably the trickiest part of creating a fully XFS system is migrating the / filesystem since that is the system that supports the entire rest of the system and it cannot actually be unmounted while the system is running. If you have extra partitions that can be mounted as / then you will be able to do it something like this(I am using /dev/hda4 as the spare partition and /dev/hda2 as / for this example):

```
$ mkfs −t ext2 /dev/hda4
$ mkdir /mnt/temp
$ mount −t ext2 /dev/hda4 /mnt/temp
$ cd /
$ tar lcf − .|(cd /mnt/temp; tar xpvf − )
```

Notice I have used tar here to copy the files from the / fs to the spare partition. Alternatively you could use cp −dpR, but if you use tar like I've shown here with the −l flag it will copy only files from within the / fs (i.e. if you have another partition mounted as /usr it won't copy those).

The next step will be to change all references to /dev/hda2 to /dev/hda4 in /etc/fstab and in /etc/lilo.conf and run lilo. You'll then need to reboot the system again.

After rebooting the system /dev/hda4 will be mounted as / and your original / filesystem (/dev/hda2) will not be mounted. You can now create the new XFS filesystem on /dev/hda2.

```
$ mkfs −t xfs /dev/hda2
```

Then mount the new xfs fs:

```
$ mount −t xfs /dev/hda2 /mnt/temp
```

And copy the original / fs back to its original home:

```
$ cd /
$ tar lcf − .|(cd /mnt/temp; tar xpvf −)
```

Once again you will need to change all instances of /dev/hda4 in /etc/fstab and /etc/lilo.conf and run lilo. You will also need to change the filesystem type for / in /etc/fstab. It should now look something like this:

```
/dev/hda2               /                       xfs    defaults 1 1
```

Note

On some linux distributions the options given to the out−of−box fstab may be more in depth than just "defaults." For instance, on Debian systems they use "defaults,errors=remount−ro." The mount options are different for every filesystem with the exception of the keyword "defaults." Unless you know the specific XFS mount options you want you should stick with just the defaults option. In the Debian example given, the errors option is not available with XFS and will cause your filesystem not to mount.

Additionally, filesystem labels are becoming more popular so you may see an entry in your fstab that looks something like this:

```
LABEL=/                 /                       ext2    defaults        1 1
```

The easiest way to avoid problems is to simply replace the referenced label with the proper device file name (i.e. if /dev/hda1 is labeled / replace "LABEL=/" with "/dev/hda1").

Now reboot the system with the new xfs / filesystem.

Of course there are a lot of other ways to accomplish the root filesystem migration and if you think you have a good one I would definitely like to hear it and will put it in this doc if it seems like a simpler way than what is here. I, myself, didn't have a spare partition to work with but had a CD burner so I burnt a cd of my root filesystem to mount as root while I created the new xfs /. In all cases, however, the basic commands for creating and mounting the new filesystem will be the same.

4.2. Finishing up

The last of the process is fairly simple and essentially the same process of swapping around partitions while making new filesystems as was done for /. I recommend that you do the rest of this process with the system in single user mode so you can unmount everything other than / and do all of the swapping without having to reboot a million times. You can boot to single user mode by either issueing a runlevel change command to the init process like so:

```
$ telinit 1
```

or by rebooting and asking for single user mode at the Lilo prompt:

```
LILO Boot: xfs single
```

This will boot the system and drop you into a root shell with no outside connections or virtual terminals so there is no chance of any of the filesystems being in use by other users or processes (causing them to be busy so you can't unmount them). Now you can mount the spare partition, as before, copy one of the remaining filesystems to be migrated onto it (you will probably have to remove the existing contents leftover from /), unmount the old filesystem, create the xfs filesystem on it, remount it as xfs, and copy the old filesystem back onto it. Lets say you have a /dev/hda3 partition mounted as /usr. The process would go something like this:

```
$ mount −t ext2 /dev/hda4 /mnt/temp
$ cd /usr
$ tar lcf − .|(cd /mnt/temp; tar xpvf − )
$ cd /mnt/temp
$ umount /usr
$ mkfs −t xfs /dev/hda3
$ mount −t xfs /dev/hda3 /usr
$ tar lcf − .|(cd /usr; tar xpvf − )
```

Don't forget to change the filesystem type in /etc/fstab for /usr to xfs.

That's all there is to it. The rest of the filesystems to be migrated will work the same way, then you can reboot to full miltiuser mode and you've got your "Linux on Steroids!"